

A METHOD FOR ANALYSIS OF C^1 -CONTINUITY OF SUBDIVISION SURFACES

Denis Zorin

Technical Report No. CSL-TR-98-764

May 1998

The research was supported in part through grants from Pixar, the Intel Corporation, Microsoft, the Charles Lee Powell Foundation, the Sloan Foundation, an NSF CAREER award (ASC-96-24957), the NSF STC for Computer Graphics and Scientific Visualization (ASC-89-20219), the Office of the Director, Defense Research and Engineering, the Air Force Office of Scientific Research (F49620-96-1-0471), as part of the MURI program.

**A Method for Analysis Of C^1 -Continuity
of Subdivision Surfaces
Denis Zorin
Technical Report No. CSL-TR-98-764
May 1998**

A sufficient condition for C^1 -continuity of subdivision surfaces was proposed by Reif [17] and extended to a more general setting in [22]. In both cases, the analysis of C^1 -continuity is reduced to establishing injectivity and regularity of a characteristic map. In all known proofs of C^1 -continuity, explicit representation of the limit surface on an annular region was used to establish regularity, and a variety of relatively complex techniques were used to establish injectivity. We propose a new approach to this problem: we show that for a general class of subdivision schemes, regularity can be inferred from the properties of a sufficiently close linear approximation, and injectivity can be verified by computing the index of a curve. An additional advantage of our approach is that it allows us to prove C^1 -continuity for all valences of vertices, rather than for an arbitrarily large, but finite number of valences. As an application, we use our method to analyze C^1 -continuity of most stationary subdivision schemes known to us, including interpolating Butterfly and Modified Butterfly schemes, as well as the Kobbelt's interpolating scheme for quadrilateral meshes.

Keywords and phrases: Subdivision surfaces, arbitrary meshes, interval arithmetics.

Copyright ©1998
Denis Zorin

1 Introduction

Subdivision is becoming increasingly popular as a surface representation in computer graphics applications. To ensure that a subdivision algorithm has the desired behavior for almost all input data, a theoretical analysis of the surface has to be performed. For subdivision on arbitrary meshes, even the analysis of the basic property of the surfaces, C^1 -continuity, poses a considerable challenge; [18, 9, 14, 15, 19]. In this paper we describe a set of theoretical results and algorithms that make it possible to perform the C^1 -continuity tests automatically.

The principal result allowing one to analyze C^1 -continuity of most subdivision schemes, is the sufficient condition of Reif [17]. This condition reduces the analysis of stationary subdivision to the analysis of a single map, called the *characteristic map*, for each valence of vertices in the mesh. The analysis of C^1 -continuity is performed in three steps for each valence:

1. compute the control net of the characteristic map;
2. prove that the characteristic map is regular;
3. prove that the characteristic map is injective.

This map can be expressed in a closed form for spline-based subdivision schemes, such as Loop, Catmull-Clark and Doo-Sabin. For these schemes, proving regularity of the characteristic map is tedious but straightforward, as the Jacobian of the map can be expressed in terms of piecewise polynomial basis functions. Proving injectivity is somewhat more difficult. [18, 14].

Our goal is to verify regularity and injectivity automatically for arbitrary (not necessarily spline-based) subdivision schemes, once the control net for the characteristic map is known. Our approach has two additional benefits:

- With some mild assumptions on the dependence of the coefficients of the characteristic scheme on the valence, we are able to analyze C^1 -continuity for *all* valences.
- Stability of C^1 -continuity with respect to perturbations of coefficients can be estimated.

Our method is based on two results, discussed in Sections 3 and 4. The estimates of Section 3 allow us to infer regularity from the properties of the linear approximations to the limit map, which can be computed explicitly. In Section 4 we show that a regular characteristic map can be proved to be injective simply by verifying that the index of the restriction of the map to the boundary of its domain is 1. The latter result follows from self-similarity of the characteristic map: in general, it is not true that a map is injective if it is regular on its domain even if the domain is the plane and the map is polynomial¹. Computing the index of a curve is a simple procedure that can be implemented robustly.

A crucial element of our technique is the interval computation: although in many cases all required calculations can be performed symbolically, it is much more efficient and, in fact, simpler, to obtain guaranteed bounds on the quantities of interest using interval arithmetics. As an additional benefit, we are able to prove facts not about single characteristic maps defined by exact values of the control points, but about families of maps, corresponding to the control points with interval components.

Using our method, we analyze interpolating triangular and quadrilateral subdivision schemes – the Butterfly [6], the Modified Butterfly [21] and the Kobbelt schemes [12]. We also repeat the analysis for two schemes that were analyzed previously by other authors: the Loop [18] scheme and the Catmull-Clark scheme [14]. For the latter schemes we extend the analysis to all valences. It is important to emphasize that once the control points for the characteristic maps are computed, the same code is used to analyze all these schemes.

Related work. This work further extends the results presented in [19]. To the best of our knowledge, all schemes that were analyzed by other authors admitted closed-form expressions for the characteristic maps; our method for establishing regularity radically differs from symbolic methods used in [18, 9, 14]. Initial Discrete Fourier Transform (DFT) analysis that we use to find the control points for the characteristic map follows the well-established pattern used in [1, 9, 18, 20, 14]. Methods for proving injectivity of the characteristic map for spline-based invariant schemes were proposed in [18] and [14].

¹This statement is known as the Jacobian conjecture for dimension 2, and a counterexample was found by S. Pinchuk in [16].

Our estimates of the errors of linear approximations rely on the work of Cavaretta, Dahmen and Micchelli [2], and on the work of Cohen, Dyn and Levin [3] on matrix subdivision.

Finally, we extensively use interval arithmetics (see, for example, [13]).

Overview. In Section 2, we describe the notation for subdivision on complexes and state relevant results from [22] and [19]. In Section 3, we discuss the basic properties of the matrix subdivision schemes, and derive estimates for the convergence rates of linear and piecewise constant approximations to the limit functions generated by subdivision. These estimates apply to the surfaces as well as the directional derivatives of the coordinate functions.

In Section 4, we prove that the index of a curve can be used to test injectivity of the characteristic map. Section 5 provides a brief description of algorithms for verification of C^1 -continuity based on the results of Sections 3 and 4.

Section 6 describes the DFT analysis specific to invariant schemes. C^1 -continuity of the Butterfly and the Modified Butterfly schemes is analyzed in Section 7; the Kobbelt interpolating scheme for quadrilateral meshes is considered in Section 8.

In Section 9 we briefly discuss the applications of our method to other schemes.

2 Subdivision Schemes

In this section we summarize the main definitions and facts about subdivision on complexes that we use. The main result is the generalization of Reif's sufficient condition (Theorem 2.1. More details can be found in [22, 19].

2.1 Subdivision on Complexes

Simplicial complexes. Subdivision surfaces are naturally defined as functions on two-dimensional simplicial complexes. Recall that a simplicial complex K is a set of vertices, edges and triangles in \mathbf{R}^N , such that for any triangle all its edges are in K , and for any edge its vertices are in K . We assume that there are no isolated vertices or edges. $|K|$ denotes the union of triangles of the complex regarded as a subset of \mathbf{R}^N with induced metric. We say that two complexes K_1 and K_2 are *isomorphic* if there is a homeomorphism between $|K_1|$ and $|K_2|$ that maps vertices to vertices, edges to edges and triangles to triangles.

A *subcomplex* of a complex K is a subset of K that is a complex. A 1-neighborhood $N_1(v, K)$ of a vertex v in a complex K is the subcomplex formed by all triangles that have v as a vertex. The m -neighborhood of a vertex v is defined recursively as a union of all 1-neighborhoods of vertices in the $(m - 1)$ -neighborhood of v . We omit K in the notation for neighborhoods when it is clear what complex we refer to.

Recall that a *link* of a vertex is the set of edges of $N_1(v, K)$ that do not contain v . We consider only complexes with all vertices having links that are connected simple polygonal lines, open or closed. If the link of a vertex is an open polygonal line, this vertex is a boundary vertex, otherwise it is an internal vertex.

Most of our constructions use two special types of complexes — *k-regular complexes* \mathcal{R}_k and the *regular complex* \mathcal{R} . Each complex is simply a triangulation of the plane consisting of identical triangles. In the regular complex each vertex has exactly 6 neighbors. In a k -regular complex all vertices have 6 neighbors, except one vertex C , which has k neighbors. We call C the central vertex of a k -regular complex and identify it with zero in the plane.

Subdivision of simplicial complexes. We can construct a new complex $D(K)$ from a complex K by subdivision, adding a new vertex for each edge of the complex and replacing each old triangle with four new triangles. Note that k -regular complexes are self-similar, that is, $D(\mathcal{R}_k)$ and \mathcal{R}_k are isomorphic.

We use notation K^j for j times subdivided complex $D^j(K)$ and V^j for the set of vertices of K^j . Note that the sets of vertices are nested: $V^0 \subset V^1 \subset \dots$

Subdivision schemes. Next, we attach values to the vertices of the complex; in other words, we consider the space of functions $V \rightarrow B$, where B is a vector space over \mathbf{R} . The range B is typically \mathbf{R}^l or \mathbf{C}^l for some l . We denote this space $\mathcal{P}(V, B)$, or $\mathcal{P}(V)$, if the choice of B is not important.

A *subdivision scheme* for any function $p^j(v)$ on vertices V^j of the complex K^j computes a function $p^{j+1}(v)$ on the vertices of the subdivided complex $D(K) = K^1$. More formally, a subdivision scheme is a collection of operators $S[K]$ defined for every complex K , mapping $\mathcal{P}(K)$ to $\mathcal{P}(K^1)$. We consider only subdivision schemes that are linear, that is, the operators $S[K]$ are linear functions on $\mathcal{P}(K)$. In this case the subdivision operators are defined by equations

$$p^1(v) = \sum_{w \in V} a_{vw} p^0(w)$$

for all $v \in V^1$. The coefficients a_{vw} may depend on K .

We restrict our attention to subdivision schemes which are finitely supported, locally invariant with respect to a set of isomorphisms of complexes and affinely invariant.

A subdivision scheme is *finitely supported* if there is an integer M such that $a_{vw} \neq 0$ only if $w \in N_M(v, K)$ for any complex K (note that the neighborhood is taken in the complex K^{j+1}). We call the minimal possible M the *support size* of the scheme.

We assume our schemes to be *locally defined* and *invariant with respect to isomorphisms of complexes*².

Together these two requirements can be defined as follows: there is a constant L such that if for two complexes K_1 and K_2 and two vertices $v_1 \in V_1$ and $v_2 \in V_2$ there is an isomorphism $\rho : N_L(v_1, K_1) \rightarrow N_L(v_2, K_2)$, such that $\rho(v_1) = v_2$, then $a_{v_1 w} = a_{v_2 \rho(w)}$. In most cases, the *localization size* $L = M$.

The final requirement that we impose on subdivision schemes is *affine invariance*: if T is a linear transformation $B \rightarrow B$, then for any v $T p^{j+1}(v) = \sum a_{vw} T p^j(v)$. This is equivalent to requiring that all coefficients a_{vw} for a fixed v sum up to 1.

Limit functions. For each vertex $v \in \cup_{j=0}^{\infty} V^j$ there is a sequence of values $p^i(v)$, $p^{i+1}(v)$, ... where i is the minimal number such that V^i contains v .

Definition 2.1. A subdivision scheme is called *convergent* on a complex K , if for any function $p \in \mathcal{P}(K, B)$ there is a continuous function f defined on $|K|$ with values in B , such that

$$\lim_{j \rightarrow \infty} \sup_{v \in V^j} \|p^j(v) - f(v)\|_2 \rightarrow 0$$

The function f is called the *limit function* of subdivision.

Notation: $f[p]$ is the limit function generated by subdivision from the initial values $p \in \mathcal{P}(K)$.

It is easy to show that if a limit function exists, it is unique. A *subdivision surface* is the limit function of subdivision on a complex K with values in \mathbf{R}^3 . In this case we call the initial values $p^0(v)$ the *control points* of the surface.

Locally any surface generated by a subdivision scheme on an arbitrary complex can be thought of as a part of a subdivision surface defined on a k -regular complex. Note that this fact alone does not guarantee that it is sufficient to study subdivision schemes only on k -regular complexes (see [22]). If the number of control points of the initial complex for a k -gonal patch is less than the number of control points of the central k -gonal patch in the k -regular complex, then only a proper subspace of all possible configurations of control points on the subdivided complexes can be realized. Although it is unlikely, it is possible that for such complexes almost all configurations of control points will lead to non-smooth surfaces, while the scheme is smooth on the k -regular complexes.

Subdivision matrices. Consider the part of a subdivision surface $f[y]$ with $y \in U_1^j = |N_1(0, \mathcal{R}_k^j)|$, defined on the k -gon formed by triangles of the subdivided complex \mathcal{R}_k^j adjacent to the central vertex. It is straightforward to show that the values at all dyadic points in this k -gon can be computed given the initial values $p^j(v)$ for $v \in N_L(0, \mathcal{R}_k^j)$. In particular, the control points $p^{j+1}(v)$ for $v \in N_L(0, \mathcal{R}_k^{j+1})$ can be computed using only control points $p^j(w)$ for $w \in N_L(0, \mathcal{R}_k^j)$. Let \bar{p}^j be the vector of control points $p^j(v)$ for $v \in N_L(0, \mathcal{R}_k^j)$. Let $p+1$ be the number of vertices in $N_L(0, \mathcal{R}_k)$.

As the subdivision operators are linear, \bar{p}^{j+1} can be computed from \bar{p}^j using a $(p+1) \times (p+1)$ matrix S^j : $\bar{p}^{j+1} = S^j \bar{p}^j$

If for some m and for all $j > m$, $S^j = S^m = S$, we say that the subdivision scheme is *stationary on the k -regular complex*, or simply *stationary*, and call S the *subdivision matrix* of the scheme. Note that our definition in the case $k = 6$ is weaker than the standard definition of stationary schemes on regular complexes [2].

As we will see, eigenvalues and eigenvectors of the matrix have fundamental importance for smoothness of subdivision.

²In fact, we only need invariance with respect to sufficiently large set of isomorphisms of complexes; this allows us to include schemes defined on tagged complexes; see [22, 19].

Eigenbasis functions. let $\lambda_0 = 1, \lambda_i, \dots, \lambda_J$ be different eigenvalues of the subdivision matrix in nonincreasing order, the condition $\lambda_0 > \lambda_1$ is necessary for convergence.

For any λ_i let $J_j^i, j = 1 \dots$ be the complex cyclic subspaces corresponding to this eigenvalue.

Let n_j^i be the *orders* of these cyclic subspaces; the order of a cyclic subspace is equal to its dimension minus one.

Let $b_{jr}^i, r = 0 \dots n_j^i$ be the complex generalized eigenvectors corresponding to the cyclic subspace J_j^i . The vectors b_{jr}^i satisfy

$$Sb_{jr}^i = \lambda_i b_{jr}^i + b_{j,r-1}^i \quad \text{if } r > 0, \quad Sb_{j0}^i = \lambda_i b_{j0}^i \quad (2.1)$$

The complex *eigenbasis functions* are the limit functions defined by $f_{jr}^i = f[b_{jr}^i] : U_1 \rightarrow \mathbf{C}$

Any subdivision surface $f[p] : U_1 \rightarrow \mathbf{R}^3$ can be represented as

$$f[p](y) = \sum_{i,j,r} \beta_{jr}^i f_{jr}^i(y) \quad (2.2)$$

where $\beta_{jr}^i \in \mathbf{C}^3$, and if $b_{jr}^i = \overline{b_{lr}^k}, \beta_{jr}^i = \overline{\beta_{lr}^k}$, where the bar denotes complex conjugation.

One can show using the definition of limit functions of subdivision and (2.1) that the eigenbasis functions satisfy the following set of *scaling relations*:

$$f_{jr}^i(y/2) = \lambda_i f_{jr}^i(y) + f_{j,r-1}^i(y) \quad \text{if } r > 0, \quad f_{j0}^i(y/2) = \lambda_i f_{j0}^i(y) \quad (2.3)$$

C^1 -continuity of surfaces. By C^1 -continuous surfaces we mean two-dimensional manifolds immersed (not necessarily embedded) in \mathbf{R}^3 (see [22] for more detailed discussion). It can be easily shown that no scheme can generate C^1 -continuous surfaces for any configuration of control points. Hence, we only require that subdivision generates C^1 -continuous surfaces for any choice of control points on a complex K , except a nowhere dense set of configurations. In almost all cases, for local schemes C^1 -continuity for arbitrary complexes follows from C^1 -continuity on k -regular complexes. A subtle problem may occur, however, for *constrained* complexes (see [22] for further details).

Characteristic maps.

Definition 2.2. The characteristic map $\Phi : U_1 \rightarrow \mathbf{R}^2$ is defined for a pair of cyclic subspaces J_b^a, J_d^c of the subdivision matrix as (f_{a0}, f_{a1}) if $J_b^a = J_d^c, \lambda_a$ is real, (f_{a0}, f_{c0}) if $J_b^a \neq J_d^c, \lambda_a, \lambda_c$ are real, and $(\Re f_{a0}, \Im f_{a0})$ if $\lambda_a = \bar{\lambda}_c, b = d$.

Three types of characteristic maps are shown in Figure 1.

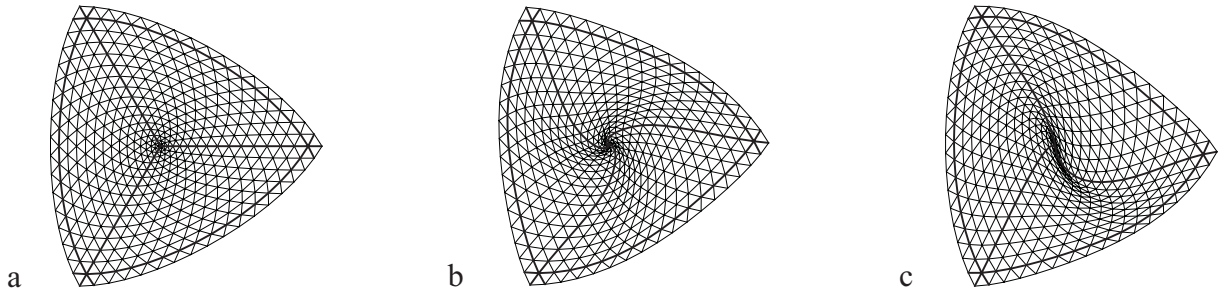


Figure 1: Three types of characteristic maps: control points after 4 subdivision steps are shown. a. Two real eigenvalues. b. A pair of complex-conjugate eigenvalues. c. single eigenvalue with Jordan block of size 2.

The domain of a characteristic map is the k -gon U_1 , consisting of k triangles of the regular complex; we call these triangles *segments*. We assume that the subdivision scheme generates C^1 -continuous limit functions the regular complexes, and the characteristic map is C^1 -continuous inside each segment and has continuous one-sided derivatives on the boundary.

Sufficient condition for C^1 -continuity. The following sufficient condition is a special case of the condition that was proved in [22]. Although all our constructions apply in the more general case, we state the only a simplified version of the criterion to simplify the presentation. This form captures the main idea of the sufficient condition. This condition generalizes Reif's condition [17].

Define for any two cyclic subspaces $\text{ord}(J_j^i, J_l^k)$ to be $n_j^i + n_l^k$, if $J_j^i \neq J_l^k$; let $\text{ord}(J_j^i, J_j^i) = 2n_j^i - 2$; note that for $n_j^i = 0$, this is a negative number, and it is less than ord for any other pair. This number allows us to determine which components of the limit surface contribute to the limit normal (see [22, 19] for details). We say that a pair of cyclic subspaces J_b^a, J_d^c is *dominant* if for any other pair J_j^i, J_l^k we have either $|\lambda_a \lambda_c| > |\lambda_i \lambda_k|$, or $|\lambda_a \lambda_c| = |\lambda_i \lambda_k|$ and $\text{ord}(J_b^a, J_d^c) > \text{ord}(J_j^i, J_l^k)$. Note that the blocks of the dominant pair may coincide.

Theorem 2.1. *Let b_{jr}^i be a basis in which a subdivision matrix S has Jordan normal form. Suppose that there is a dominant pair of J_b^a, J_d^c . If $\lambda_a \lambda_c$ positive real, and the Jacobian of the characteristic map of J_b^a, J_d^c has constant sign everywhere on U_1 except zero, then the subdivision scheme is tangent plane continuous on the k -regular complex.*

If the characteristic map is injective, the subdivision scheme is C^1 -continuous.

In the special case when all Jordan blocks are trivial, this condition reduces to an analog of the Reif's condition.

To apply Theorem 2.1, we use self-similarity of the characteristic map: for any $t \in U_1$, the Jacobian $J[\Phi](t/2) = 4\lambda_a \lambda_b [\Phi](t)$. It is immediately clear that to prove regularity of the characteristic map it is sufficient to consider the Jacobian on a single annular portion of U_1 as shown in Figure 2. To prove injectivity, we use the criterion described in Section 4.

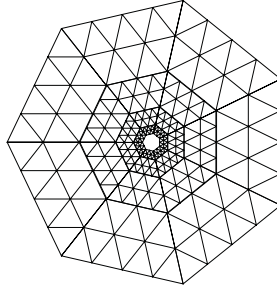


Figure 2: The k -gon without origin $U_1 \setminus \{0\}$ can be decomposed into similar rings, each two times smaller than the previous ring. The size of the ring is chosen in such a way that the control set of any ring does not contain the extraordinary vertex. In this figure the control set is assumed to consist out of the vertices of the triangles of the ring itself, and of a single layer of vertices outside the ring.

Our goal is to develop an efficient general method that would allow us to apply this condition to arbitrary subdivision schemes. In the next two sections we develop theoretical foundation for application of this criterion. In Section 3, we prove that regularity of the characteristic map can be verified using linear approximations to the map. This is sufficient to analyze tangent plane continuity. In Section 4, we show that injectivity of the characteristic map can be verified by computing the index of a curve.

3 Approximation Errors of Stationary Subdivision

We have observed that regularity of the characteristic map can be established, if it is known that the scheme is regular on an annular region (ring) shown in Figure 2. All control vertices for a layer are regular, and the subdivision rules that are used to compute the limit surface on the ring are the rules used for the regular complex. Clearly, the ring cannot be identified with a subset of a regular complex. However, such identification can be done for each of the k segments of the ring together with its control points. Therefore, if we can prove regularity of a limit map on the regular complex, we can apply the same algorithm to prove regularity of the characteristic map for each segment.

Our method is based on the observation that we can define a subdivision scheme for the vector of differences. The limit function of this scheme is the vector of partial derivatives of the characteristic map. We estimate the error of the

piecewise-linear approximations produced by this scheme. From linear approximations and errors we compute upper and lower bounds for the Jacobian of the characteristic map. If these bounds have the same sign, we can conclude that the map is regular.

Our derivations are similar to the derivations in Chapters 2 and 3 of Cavaretta, Dahmen and Micchelli [2], and those found in Dyn, Levin and Micchelli [7]. We have to consider convergence not only of the scheme, but also of the corresponding scheme for differences, which in general is a *matrix subdivision scheme*. For this reason, some of the theorems in [2] have to be generalized to the matrix case. Cohen, Dyn and Levin have developed the basic theory of univariate matrix schemes in [3]. We use *multivariate matrix subdivision schemes*, that is, we need a synthesis of the theories presented in [2] and [3]. The theory of matrix subdivision differs from the theory of scalar subdivision in a non-trivial way, when the components of the limit functions generated by the scheme are interdependent [3]. However, this case is of little interest to us: if the components of the difference scheme are interdependent, the limit surfaces are degenerate. Hence we can assume independence of components. With this assumption, the results in [2] can be readily extended to the matrix case.

Definitions. For a regular complex, the vertices can be identified with the integer points in the plane. In general, we can consider functions on the integer lattice \mathbf{Z}^2 in \mathbf{R}^2 . Most of the discussion applies to integer lattices \mathbf{Z}^s of arbitrary dimension with minor changes. We perform the derivations for the case $s = 2$ to simplify the presentation. We use Greek letters to denote multiindices corresponding to the points of the lattice: $\alpha = (\alpha_1, \alpha_2)$. A stationary matrix subdivision scheme on a regular complex is defined by the equation

$$(Sp)(v_\alpha) = \sum_{\beta} A_{\alpha-2\beta} p(v_\beta)$$

where A_α are $n \times n$ matrices, and p_α are in $(\ell_2^\infty)^n = (\ell^\infty(\mathbf{Z}^2) \times \dots \times \ell^\infty(\mathbf{Z}^2))$, the space of 2-dimensional sequences of n -dimensional vectors with bounded norm. As we are interested in schemes with finite support, all results can be extended to arbitrary vectors p_α in a straightforward manner (see [2]). We are primarily interested in the cases $n = 1, 2, 4$, corresponding to scalar subdivision, difference schemes and second difference schemes respectively.

If a subdivision scheme converges on the regular complex, there is a matrix function $\Phi : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}$, such that any limit function $f[p]$ generated from the initial values $p \in (\ell_2^\infty)^n$, can be written as $f[p](t) = \sum_{\alpha} \Phi(t - \alpha) p_\alpha$, where $t \in \mathbf{R}^2$. The matrix refinable function Φ satisfies the refinement relation

$$\Phi(t) = \Phi(2t - \alpha) A_\alpha$$

The function Φ can be obtained as a limit of subdivision applied to initial matrix data Δ with $\Delta_\alpha = 0$ if $\alpha \neq 0$, and Δ_0 is $n \times n$ identity matrix.

We say that a matrix scheme is *nondegenerate* if the vectors $f[p](t)$, $t \in \mathbf{R}^2$, for all t for some $p \in (\ell_2^\infty)^n$ span the whole space \mathbf{R}^n . It is straightforward to show, following the derivation in [2], Proposition 2.1, that for a nondegenerate matrix scheme to be convergent the following condition is necessary:

Theorem 3.1. *For a nondegenerate matrix scheme S to be convergent it is necessary that for any $e = (e_1, e_2)$, $e_i \in \{0, 1\}$*

$$\sum_{\alpha} A_{e-2\alpha} = I \tag{3.1}$$

where I is the identity matrix.

A matrix subdivision scheme is *stable* (or, more precisely, L^∞ -stable) [3] if the matrix refinable function Φ corresponding to the scheme satisfies the inequalities

$$c_1 \sup_{\alpha} \|p\|_{\infty} \leq \sup_{t \in \mathbf{R}^2} \left\| \sum_{\alpha} \Phi(t - \alpha) p \right\|_{\infty} \leq c_2 \sup_{\alpha} \|p\|_{\infty} \tag{3.2}$$

for some positive constants c_1, c_2 and any $p \in (\ell_2^\infty)^n$.

Convergence condition. To analyze convergence of matrix subdivision schemes, we use the *contraction functions*. Let $D(p)$ be a real-valued nonnegative function defined on $(\ell_2^\infty)^n$. A subdivision scheme S is *contractive* relative to D if there is an integer N and a positive constant $\gamma_N < 1$, for which we have

$$D(S^N p) \leq \gamma_N D(p) \quad \text{for any } p \in (\ell_2^\infty)^n$$

A typical contraction function has the form $\|\nabla p\|_\infty$, where ∇p denotes the vector of directional differences.

The following theorem is a direct generalization of of Theorem 3.1 [2], with the proof extended without any changes from the scalar case.

Theorem 3.2. *Let S be a matrix subdivision scheme, and D a contraction function. Suppose that for some scheme B , which we call a comparison scheme,*

$$\|Sp - Bp\|_\infty \leq cD(p) \quad \text{for any } p \in (\ell_2^\infty)^n$$

where c is a constant.

If the comparison scheme is stable and converges, then S also converges.

We use three types of comparison schemes in our analysis: schemes that produce piecewise constant, piecewise linear, and piecewise bilinear limit functions.

Error estimates. Using Theorem 3.2, we can derive error estimates for the piecewise-linear, bilinear or constant approximations of the limit function of subdivision. Let L^m be the limit function obtained by applying the comparison scheme B to the control values p^m . This is our approximation. The choice of B guarantees that the limit functions of B can be computed trivially. Then we have

$$\|L^{m+1} - L^m\|_\infty = \|(S - B)p^m\|_\infty < cD(p^m)$$

Suppose that $m = kN + q$; then $D(p^m) < \gamma^k D(p^q)$ and

$$\begin{aligned} \|L^\infty - L^m\|_\infty &\leq \sum_{j=0}^{\infty} \|L^{m+j+1} - L^{m+j}\|_\infty = \\ &= \sum_{j=0}^{\infty} \|(S - B)p^{m+j}\|_\infty \leq c \sum_{j=0}^{\infty} D(p^{m+j}) = \\ &= c \sum_{i=1}^{\infty} \sum_{q=1}^N D(p^{m+iN-q}) = c \sum_{i=1}^{\infty} \left(\sum_{q=1}^{N-1} D(p^{m+iN-q}) + D(p^{m+(i-1)N}) \right) \end{aligned}$$

If $m \geq N$,

$$\|L^\infty - L^m\|_\infty \leq \frac{c}{1-\gamma} \left(\gamma \sum_{q=1}^{N-1} D(p^{m-q}) + D(p^m) \right) \quad (3.3)$$

Estimating c and γ . To use the estimates (3.3), we need to compute the constants γ_N and c . These constants clearly depend on the choice of the contraction function and on the choice of the comparison scheme. We use the contraction functions of the form $\|\nabla p\|_\infty$, where $\nabla : (\ell_2^\infty)^n \rightarrow (\ell_2^\infty)^{2n}$ is a difference operator, which assigns to each vector p of length n the vector of differences in 2 independent directions of length $2n$. Specific choice of ∇ can be adapted to the scheme; the simplest choice is $[(\nabla p)_{2m+i}]_\alpha^m = [p_{\alpha+e_i}]^m - [p_\alpha]^m = \Delta_{e_i}[p_\alpha]$, where e_i , $i = 0, 1$, is one of the multiindices $(0, 1)$ and $(1, 0)$. The superscript m of p denotes the component of p , $m = 1 \dots n$. Other possible choices include replacing vectors e_i by displacements in other directions. Specific choices are discussed in Sections 7-9. Whenever for a scheme S there is a difference scheme S' satisfying the commutation formula

$$\nabla S p = S' \nabla p \quad (3.4)$$

for any p . If $\|S'^N\|_\infty < 1$ for some N , we can use $\|\nabla p\|_\infty$ as a contraction function, because $\|\nabla S^N p\|_\infty \leq \|S'^N\|_\infty \|\nabla p\|_\infty$. While in most cases the simplest choice of ∇ is theoretically possible, in practice the constant N can be quite large. In certain cases, such as the Butterfly scheme, different contraction functions yield better results.

For contraction functions of the type described above computation of γ_N and is reduced to computing the (matrix) difference scheme S' and its sup norm. Lemma 2.3 from [2], which directly generalizes to the matrix case, yields an algorithm for computing the matrix Laurent polynomial of the difference scheme.

Formulas for γ . We describe the algorithm for computing the difference scheme in the bivariate case, for the simplest comparison function $D(p) = \|\nabla p\|_\infty$, $p \in (\ell^\infty)^n$, with ∇p described above. In a more explicit form,

$$D(p) = \sup_{\alpha} \max_m ([\Delta_{(1,0)} p \alpha]^m, [\Delta_{(0,1)} p \alpha]^m)$$

Let $A(z)$ be the Laurent polynomial with matrix coefficients of a matrix scheme S , with $z = (z_1, z_2)$. The commutation formula in the z -domain can be written as

$$\begin{pmatrix} (z_1 - 1)A(z)p(z) \\ (z_2 - 1)A(z)p(z) \end{pmatrix} = A'(z) \begin{pmatrix} z_1^2 - 1 \\ z_2^2 - 1 \end{pmatrix} = \begin{pmatrix} T_{11}(z)(z_1^2 - 1) + T_{12}(z)(z_2^2 - 1) \\ T_{21}(z)(z_1^2 - 1) + T_{22}(z)(z_2^2 - 1) \end{pmatrix} \quad (3.5)$$

for any p . The matrix Laurent polynomial A' with the coefficients of size $2n \times 2n$ is the Laurent polynomial of the difference scheme S' . Each coefficient of A' is composed out of the coefficients of $n \times n$ blocks $T_{ij}(z)$. The blocks T_{ij} are not defined uniquely, and can be computed in a variety of ways. A method suggested by Lemma 2.3 of [2], yields the following formulas for a decomposition $R(z) = (z_1^2 - 1)T_1(z) + (z_2^2 - 1)T_2(z)$

$$\begin{aligned} T_1(z) &= \frac{(1 - z_2)R(z_1, -1) + (1 + z_2)R(z_1, 1)}{2(z_1^2 - 1)} \\ T_2(z) &= \frac{2R(z) - (1 - z_2)R(z_2, -1) - (1 + z_2)R(z_1, 1)}{2(z_2^2 - 1)} \end{aligned} \quad (3.6)$$

Whenever a scheme with the Laurent polynomial A satisfies Theorem 3.1, it follows that $A(-1, -1) = A(1, -1) = A(-1, 1) = 0$. Then $T_1(z)$ and $T_2(z)$ are guaranteed to be matrix Laurent polynomials, rather than rational functions, for $R(z) = (z_i - 1)A(z)$, $i = 1, 2$.

These formulas can be used to compute the blocks for each line in (3.5). Note that the formulas are asymmetric, and care must be taken to choose the order of variables z_1 and z_2 to obtain better estimates. The rule of thumb is that the norm of the off-diagonal blocks should be small; then for schemes with factorizable Laurent polynomials the $2n \times 2n$ difference scheme can be decomposed into two $n \times n$ schemes.

The same method can be used to compute difference schemes for other choices of ∇ that we use. Once S' is known, we compute γ_N as $\|S'^N\|_\infty = \|A(z)A(z^2) \dots A(z^{2^N})\|_\infty$ using the formula for the sup-norm of a bivariate polynomial with $n \times n$ matrix coefficients $A_{ij} = [a^m]_{ij}$:

$$\|C(z)\|_\infty = \max_{e_1, e_2 \in \{0,1\}, m=1..n} \sum_{l=1..n} \sum_{ij} |c_{2i+e_1, 2j+e_2}^{lm}| \quad (3.7)$$

Computing c . To compute c , it is sufficient to observe that if the comparison scheme B is nondegenerate and convergent, then $B(z) - A(z)$ can be always decomposed in a way similar to $R(z)$ in (3.6). This means that we can represent $B - S$ as $\tilde{S}\nabla$, for some $2n \times 2n$ matrix scheme \tilde{S} , which leads to the estimate of c as $\|\tilde{S}\|_\infty$.

Summary of the estimates. We have obtained the following estimates for the constants $\gamma_N, c, \gamma_N^D, c^D$, characterizing approximation errors for the approximations of the limit functions and its derivatives respectively. Let S is a convergent scalar subdivision scheme, satisfying commutation formula $\nabla S p = S' \nabla p$, where $p \in \ell^\infty$, and S' is a 2×2 matrix scheme. Let B be a scalar comparison scheme, and $(B - S)p = \tilde{S} \nabla p$. Then we can take $\gamma_N = \|S'^N\|_\infty$, and $c = \|\tilde{S}\|_\infty$. If the difference scheme $2S'$ converges and satisfies the commutation formula $\nabla 2S' p = S'' \nabla p$, where S'' is a 4×4 matrix subdivision scheme, $\gamma_N^D = \|S''\|_\infty$. If the difference scheme $2B'$ corresponding to a comparison scheme B is also a comparison scheme, and $(2S' - 2B')p = \tilde{S}' \nabla p$, then $c^D = \|\tilde{S}'\|_\infty$.

Limitations of the method. While for any C^1 -continuous subdivision scheme and for all operators ∇ that we use the difference schemes S' and S'' are defined, it is not guaranteed that for any N $\|S'^N\|_\infty < 1$ or $\|S''^N\|_\infty < 1$. This is the main limitation on the applicability of our method. Even for a convergent scheme it might be possible that N $\|S'^N\|_\infty > 1$ for all N . However, note that a sharper estimate can be made for γ : we are interested in the action of the difference scheme not on arbitrary elements of $(\ell_2^\infty)^{2n}$, but only on the elements that have the form ∇p for some $p \in (\ell_2^\infty)^n$. Therefore, we can use $\|S'^N|_\nabla\|_\infty$ instead of $\|S'^N\|_\infty$, where $|_\nabla$ denotes restriction to the subspace of differences. If we use this norm, then the following theorem holds:

Theorem 3.3. *A matrix subdivision scheme S is nondegenerate, convergent and stable, if and only if there is a difference scheme S' satisfying the commutation formula 3.4 and such that $\lim_{N \rightarrow \infty} \|S'^N|_\nabla\|_\infty = 0$.*

The proof of the theorem is identical to the proof of Theorem 2.3, [2]. The additional condition – stability of the scheme – is quite weak, and in all cases of interest is likely to be satisfied.

Computing the norm $\|\cdot|_\nabla\|_\infty$ is possible for schemes with finite support, but is substantially more complicated than computing the sup norm. For schemes with factorizable Laurent polynomials for suitable choices of ∇ , $\|\cdot|_\nabla\|_\infty$ is often equal to $\|\cdot\|_\infty$ (for example, this is true for tensor product schemes). However, for certain schemes with nonfactorizable polynomials computing $\|\cdot|_\nabla\|_\infty$ may be the only option.

4 Injectivity of the Characteristic Map

In general, it is difficult to establish injectivity of a map defined as a limit of a subdivision process. Even if the Jacobian of a map is nonzero everywhere, only local injectivity is guaranteed. However, the special structure of the characteristic maps allows one to reduce the injectivity test to computing the index of a curve, a relatively simple and fast operation: for example, the index can be computed counting the number of intersections of the curve with a line.

A step in this direction was made by Peters and Reif [14]. However, their method still required closed-form expression for the derivative of the characteristic map along a line, and was formulated only for schemes invariant with respect rotations of k -regular complexes (see Section 6).

The characteristic map can be extended using scaling relations to the whole plane. In the following theorem we assume that the characteristic map is defined on \mathbf{R}^2 .

Theorem 4.1. *Suppose a characteristic map $\Phi = (f_a, f_c)$ satisfies the following conditions:*

1. *the preimage $\Phi^{-1}(0)$ contains only one element, 0;*
2. *the characteristic map has a Jacobian of constant sign at all points where it is defined.*

Then the extension of the characteristic map is surjective and is a covering away from 0. In particular, if the winding number of the image $\Phi(\gamma)$ of a curve is 1, the characteristic map is injective and the scheme is C^1 -continuous.

Proof. Three cases are possible: the characteristic map is defined by a pair of real eigenvectors, by two generalized eigenvectors from the same Jordan block corresponding to a real eigenvector, or the real and imaginary parts of an eigenvector corresponding to a complex eigenvalue.

A pair of real eigenvectors. In the first case the components of the characteristic map satisfy the scaling relations of the simplest form $f_a(y/2) = \lambda_a f_a(y)$, $f_c(y/2) = \lambda_c f_c(y)$.

First, we establish the following important fact: if a characteristic map satisfies the first two conditions of the lemma, then the map is continuous at infinity.

Consider two circles of radii r and $2r$ centered at 0 in the domain of Φ . The image $\Phi(R)$ of the ring R bounded by the two circles is compact, and does not contain 0. Thus, there is a constant $M > 0$ such that for any point p in the ring $\|\Phi(p)\| \geq M$.

Consider any point p in the domain of Φ . There is a number $k \in \mathbf{Z}$ such that $2^k p$ is contained in the ring R . Thus, by scaling relations, $\|\Phi(p)\| > \min(|\lambda_a|, |\lambda_c|)^k M$. Clearly, as $\|p\| \rightarrow \infty$, $k \rightarrow \infty$, and for any C there is C' such that if $\|p\| > C'$, $\|\Phi(p)\| > C$.

Consider the stereographic map P from the plane into the sphere without one point. The map Φ corresponds to a map on the sphere: $\Phi_S = P\Phi P^{-1} : S^2 \setminus \{N\} \rightarrow S^2$, where N is the center of projection. From the continuity of

Φ at infinity it follows that if we extend the mapping by setting $\Phi_S(N) = N$, we get a continuous mapping. As we have assumed that the Jacobian of the characteristic map has constant sign where it is defined, the mapping is also a local homeomorphism away from 0. The sphere is compact, thus its image is compact, hence closed, i.e., contains its boundary. But under local homeomorphism the points on the boundary of the image can be images only of the points of the boundary of the domain. Therefore, the only points that can be contained in the boundary of the image are 0 and N . We conclude that the image has no boundary, i.e., the mapping is surjective.

Finally, for any p set $\Phi_S^{-1}(p)$ is finite: if it were not finite, it would have a limit point (S^2 is compact). As $\Phi_S^{-1}(p)$ is a discrete set for any local homeomorphism, the only limit points that it may have are 0 and N . But $\Phi(0) = 0$ and $\Phi(N) = N$, so this is impossible. We conclude that for any point p the set $\Phi_S^{-1}(p)$ is finite. As any point $y \in \Phi_S^{-1}(p)$, $p \neq 0, N$ has a neighborhood $U(y)$ such that $\Phi_S|_{U(y)}$ is a homeomorphism, then the intersection of all neighborhoods $V = \Phi_S(U(y))$ has inverse image consisting of disjoint homeomorphic images of V . This proves that Φ_S is a covering away from 0.

Two generalized eigenvectors. The case of the characteristic map generated by imaginary and real part of a complex eigenvector corresponding to a complex eigenvalue is similar to the case of two real eigenvectors; we proceed directly to the proof for the case of two generalized eigenvectors from a single Jordan block $\Phi = (f_0, f_1)$, satisfying $f_0(\frac{y}{\lambda}) = \lambda f_0(y)$ and $f_1(\frac{y}{\lambda}) = \lambda f_1(y) + f_0(y)$.

From these equations we immediately obtain

$$\Phi(2^p y) = \frac{1}{\lambda^p} \begin{pmatrix} 1 & 0 \\ 1 & -p/\lambda \end{pmatrix} \Phi(y) = \frac{1}{\lambda^p} T \Phi(y) \quad (4.1)$$

Consider the image of a circle γ of radius r centered at 0. Let $\text{Int}(\gamma)$ be the interior domain of the simple curve γ . As $\Phi^{-1}(0)$ by assumption is $\{0\}$, then 0 is an interior point of the image of $\text{Int}(\gamma)$ and there is an open disk centered at 0 of some radius r' , which is contained in $\Phi(\text{Int}(\gamma))$. For any p the image of the disk bounded by $2^p \gamma$ is determined by the equations (4.1). It can be obtained from the image of the disk bounded by γ by affine transform $\frac{1}{\lambda^p} T$ from (4.1). If a disk D_r of radius r is contained in $\Phi(\text{Int}(\gamma))$, then the interior of the ellipse $\frac{1}{\lambda^p} T D_r$ is contained in $\Phi(\text{Int}(2^p \gamma))$. We can estimate the length of the minor axis of this ellipse: it can be represented parametrically as $(\frac{r}{\lambda^p} \cos(t), \frac{r}{\lambda^p} (\sin(t) - (p/\lambda) \cos(t)))$. The square of the distance from 0 to a point on the ellipse is

$$\frac{r^2}{\lambda^{2p}} (\cos^2(t) + (\sin(t) - \frac{p}{\lambda} \cos(t))^2) = \frac{r^2}{\lambda^{2p}} (1 + \frac{p^2}{2\lambda} (\cos(2t) + 1) - \frac{p}{\lambda} \sin(2t))$$

This quantity can be estimated from below by $(r^2/\lambda^{2p}) (1 + p^2/\lambda - p/\lambda)$

As $\lambda < 1$, the length of the minor axis increases with p for sufficiently large p . We conclude that as $p \rightarrow \infty$, the image of the exterior of $2^p \gamma$ is arbitrarily far from zero, and Φ is continuous at infinity. Then the rest of the argument that was used for the case of two eigenvectors applies.

Finally, our covering is injective, if and only if the winding number of a simple curve around zero is 1. This fact can be seen by looking at the fundamental groups of the domain and the image. The assumptions guarantee that both have fundamental group \mathbf{Z} . As for a covering the fundamental group of the covering space is a subgroup of the fundamental group of the base space, with a monomorphism induced by the covering map. A simple curve around zero is the generating element of the fundamental group of the domain. Thus, the mapping of fundamental groups is an isomorphism which is necessary and sufficient for the covering mapping to be an injection, if and only if the simple curve maps to a curve homotopic to a simple curve, i.e., one with winding number 1. □

Computing the winding number. In general, we do not have a closed-form expression for any curves on the limit surface. One way to compute the winding number of a curve is to choose a sufficiently close linear approximation and compute the winding number of the approximation. The following Proposition can be easily proved (see [19] for details):

Proposition 4.2. *Let $\gamma(t)$ is a curve in the domain of Φ , L_m is a piecewise linear approximation to Φ . Suppose for some $\varepsilon \sup_t \|\Phi(\gamma(t)) - L_m(\gamma(t))\| \leq \varepsilon$, and $\inf_t \|\Phi(\gamma(t))\| \geq 2\varepsilon$. Then the winding number of $L_m(\gamma(t))$ is equal to the winding number of $\Phi(\gamma(t))$.*

As subdivision computes linear approximations to the surface, and the approximation estimates are known, we can use this proposition to compute the winding number.

5 Algorithms

In this section we describe the algorithms for verification of C^1 -continuity of subdivision near extraordinary points based on the theorems presented in Sections 3 and 4. Two algorithms are used to analyze C^1 -continuity of a scheme near an extraordinary point of a fixed valence: the first one verifies regularity, the second verifies injectivity. We give a brief description of the algorithms; more details can be found in [19]. The source code is available from the author.

We assume that the eigenvectors and eigenvalues of the subdivision matrix defining the characteristic map are known with guaranteed precision: if x is a component of an eigenvector or an eigenvalue, it is represented by a pair of exactly representable numbers $[x_d, x_u]$ such that $x_d \geq x \leq x_u$.

All calculations are performed in interval arithmetics, which makes it possible to obtain guaranteed bounds on the computed quantities, despite using finite-precision arithmetics.

5.1 Verification of C^1 -continuity for a fixed valence.

We verify C^1 -continuity by checking regularity and injectivity of the characteristic map on a ring, as described in Section 2.

First, we verify *regularity*, computing successive linear approximations to the characteristic map and using error estimates of Section 3 to estimate the range for the Jacobian. If the computed bounds are on the same side of zero, this guarantees that the Jacobian has constant sign on the domain.

To guarantee that the condition $\Phi^{-1}(0)$ of Theorem 4.1 and the assumption of Proposition 4.2 are satisfied, it is necessary to verify that the image of the characteristic map on the ring is sufficiently far from zero. The algorithm is straightforward, and we omit the detailed description.

Finally, we compute *the winding number* of the curve obtained by restricting the linear approximation to the characteristic map to the boundary of the domain. If the winding number is 1, this completes the proof that the characteristic map is injective and regular.

In the descriptions of algorithms, \tilde{G}^i denotes the control mesh of the characteristic map after i subdivision steps. The components of the control points are stored in interval representation. The numbers ϵ_i , $i = 1, 2$ are the estimates of the errors of the linear approximation to the components of the characteristic map, and ϵ_{ij} , $i, j = 1, 2$ are the estimates of the error of the approximation by divided differences to the derivatives of the characteristic map. All errors are computed using formulas from Section 3.

Regularity. Once we know the error in the approximation of the derivatives by the divided differences, we can estimate the Jacobian. Observe that the jacobian $J[\partial_1 f_1, \partial_2 f_2, \partial_2 f_1, \partial_1 f_2] = \partial_1 f_1 \partial_2 f_2 - \partial_2 f_1 \partial_1 f_2$, is a bilinear function of the derivatives. If the intervals for the derivatives are known, the Jacobian can be regarded as a bilinear function on a four-dimensional cube, and it attains its minimal/maximal value at a vertex of the cube. This leads to the following algorithm:

We present a slightly simplified version of the algorithm, which does not detect the situation when the Jacobian is guaranteed to change sign, and the test fails. A complete version can be found in [19].

```

TestRegular( $\tilde{G}^i, \gamma_D, C_D$ )

 $J_{min} := +\infty, J_{max} := +\infty$ 
foreach vertex  $\in \text{Int}(\tilde{G}^i)$ 
   $d^1 := \tilde{G}^i(i+1, j) - \tilde{G}^i(i, j)$ 
   $d^2 := \tilde{G}^i(i, j+1) - \tilde{G}^i(i, j)$ 
  compute 16 numbers  $J_l, l = 1 \dots 16$ 
  choosing signs in  $(d_1^1 \pm \epsilon_{11})(d_2^2 \pm \epsilon_{22})(d_1^1 \pm \epsilon_{12})(d_2^2 \pm \epsilon_{21})$ 
   $J_{min} := \min(J_{min}, J_l, l = 1 \dots 16)$ 
   $J_{max} := \max(J_{max}, J_l, l = 1 \dots 16)$ 
endforeach
if  $0 \notin J_{min}$  and  $0 \notin J_{max}$ 
  and  $J_{min}$  and  $J_{max}$  have the same sign
  then return true
return undefined

```

Computing the winding number. While the simplest approach to this problem is to count the number of intersections with a straight line, numerically this is not the best choice when the curve is piecewise-linear. Instead, we choose a different approach: we observe that the winding number for a piecewise-linear curve can be computed as $1/4$ of the sum of signed lengths of projections of segments onto a unit square centered at zero. As the coordinates of the vertices are represented by intervals, the actual calculation becomes somewhat more complicated. For each interval endpoint of the segment we determine the sides of the square on which the endpoint may be projected. We require the calculation to be sufficiently precise (i.e. the size of the intervals for the points to be sufficiently small) for the total number of sides intersecting the projection of the interval to be no more than two.

In the algorithm below, `head` and `tail` return the endpoints of a segment of the curve, subscripts 1 and 2 denote the coordinates, and the function `sides(x)` returns the set of sides (identified by numbers 1..4) to which a point x with interval coordinates is projected.

```

ComputeProj( $\tilde{G}^i$ )

projLength := 0
for every segment  $s$  of the curve
   $n^s := \text{head}(s)/\max(\text{head}(s)_1, \text{head}(s)_2)$ 
   $n^f := \text{tail}(s)/\max(\text{tail}(s)_1, \text{tail}(s)_2)$ 
  if  $|\text{Sides}(n^s)| > 2$  or  $|\text{Sides}(n^f)| > 2$ 
    then return fail
  intervSides :=  $\text{Sides}(n^s) \cup \text{Sides}(n^f)$ 
  if  $|\text{intervSides}| > 2$  then return undefined
  if  $1 \in \text{intervSides}$  then projLength +=  $n_2^f - n_2^s$ 
  if  $2 \in \text{intervSides}$  then projLength +=  $n_1^s - n_1^f$ 
  if  $3 \in \text{intervSides}$  then projLength +=  $n_2^s - n_2^f$ 
  if  $4 \in \text{intervSides}$  then projLength +=  $n_1^f - n_1^s$ 
endfor
return projLength

```

The algorithms described in this section are quite efficient – even in the case of the Butterfly scheme, which required 6 subdivision levels to verify C^1 -continuity of a single valence, the execution time per valence was about 7 sec on a 300MHz Pentium II.

5.2 Verification of C^1 -continuity for all valences.

The algorithms of the previous section allow us to prove that a scheme is C^1 -continuous for any given valence. We have made only weak assumptions about invariance of the schemes (rotational invariance is not required), and we have not assume any relations between the subdivision rules used near extraordinary vertices of different valences. Although one can verify C^1 -continuity for a number of valences that is sufficiently large for all practical purposes, as it was done, for example, in [18] and [14], it is not quite satisfying theoretically.

Our approach to analysis of subdivision for large valences applies to schemes invariant with respect to rotations of k -regular grids around the extraordinary vertex. In this case, the segments of the characteristic map are identical, and the analysis has to be performed for a single segment. As the valence grows, the control points of a segment approach a degenerate configuration for which all control points are on a single line. However, by rescaling the control points in one direction by $1/\sin(2\pi/k)$, where k is the valence, we typically remove the singularity. Because subdivision is affine-invariant, verifying regularity and injectivity of the rescaled characteristic map is equivalent to verifying injectivity and regularity of the original map. For all common subdivision rules, when the coefficients are defined as functions of $\cos(2\pi/k)$ and $\sin(2\pi/k)$, as k approaches infinity, the control points of the segment approach a nondegenerate limit configuration. More precisely, assume that the eigenvalues λ_a and λ_b and eigenvectors e_a and e_b defining the characteristic map are continuous functions of $c = \cos(2\pi/k)$, that can be computed in the interval form, that is, given an interval of values of c , we can compute an interval of values of λ . Recall that all algorithms that we have described operate with interval representations. Let \tilde{e}_a and \tilde{e}_b be the vectors with interval components obtained by evaluating $e_a(c)$ and $e_b(c)$ on the interval $c = [1 - \epsilon, 1]$. If we have verified that the limit map defined by \tilde{e}_a and \tilde{e}_b is injective for these interval eigenvectors, we have verified that it is injective for any valence for which $\cos(2\pi/k) > 1 - \epsilon$. Thus, we obtain a proof of C^1 -continuity for all valences greater than some k_0 at no additional cost – all we have to do is to choose the value of c to be $[1 - \epsilon, 1]$.

Finally, we observe that it is not always possible to represent an eigenvalue of the subdivision matrix as an explicit function of c . For example, for Kobbelt's scheme the characteristic polynomial has degree 6 and is not factorizable. However, the eigenvectors can always be represented as explicit functions of the eigenvalues and coefficients of the subdivision scheme. Therefore, the problem is reduced to computing the eigenvalue as a function of c with guaranteed intervals. While there are always cases when this is difficult if at all possible, in most cases it appears to be an achievable goal – see Section 8 for an example.

6 Invariant Schemes

The algorithms described in previous sections are quite general. However, specific examples of schemes considered in this paper are invariant with respect to all isomorphisms of complexes³. The algorithms of Section 5 can be further simplified and optimized for such schemes: due to symmetry, we need to consider only a single sector of the characteristic map. In addition, the transformation of the subdivision matrix described below reduces analysis of subdivision matrices to the analysis of parametric families of smaller matrices.

The constructions of this section follow the ideas of Ball and Storry [1], also used in [21] and in [14]. If a scheme is invariant with respect to all isomorphisms of complexes, in particular, it is invariant with respect to automorphisms of a k -regular complex. If ρ is an automorphism of a complex K , the coefficients of subdivision satisfy

$$a(v, w) = a(\rho(v), \rho(w)) \quad (6.1)$$

For k -regular complexes, the set of automorphisms consists of rotations around the extraordinary vertex, mirror reflections and their combinations; we use only rotations.

Let M be the localization/control size for the subdivision scheme on a k -regular complex. In this case, the control set of U_1 is a M -neighborhood of the extraordinary vertex. One sector of this neighborhood (center excluded) contains $M(M+1)/2 = N$ vertices, the total number of vertices being $Nk+1$. We will use notation $[s\ j]$ for the vertices; s is the number of the sector $s = 0 \dots k-1$, $j > 0$ is an arbitrarily chosen numbering of vertices within a sector. We use the numbering shown in Figure 3.

The central vertex is $[00]$. We assume that the numbering is chosen consistently in each sector, that is, $R^m([s\ j]) = [s + m \bmod k\ j]$ where R^m corresponds to the rotation of the plane by $2m\pi/k$.

³An example of a scheme which is *not* invariant with respect to some isomorphism is the piecewise-smooth scheme of Hoppe et al.[10]

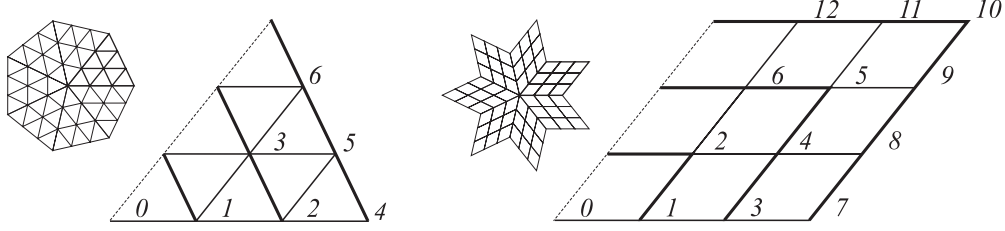


Figure 3: The numbers of the vertices in a sector of the control mesh for the characteristic map. Left: the numbering for triangular schemes; right: the numbering for quadrilateral schemes.

With this notation, (6.1) becomes $a([s'j'], [sj]) = a([(s'+m)j'], [(s+m)j])$ for any m , where the sums are modulo k .

The coefficients are functions of j, j' , and $s - s'$ only; In the cases when $j = 0$ or $j' = 0$ (one of v, w is the extraordinary vertex), the coefficients do not depend on $s - s'$. We introduce notation $a([sj], [s'j']) = a_{jj'}(s - s')$, $b_j = a([00], [sj])$, $c_j = a([sj], [00])$, $a_{00} = a([00], [00])$. The subdivision matrix will have a convenient block form if we arrange the vertices “by symmetry class”: $[0,0], [0,1], [1,1], [2,1] \dots [k-1,1], [0,2] \dots [k-1,N]$. With this ordering of vertices, the subdivision matrix has the form

$$S = \begin{pmatrix} a_{00} & \mathbf{b}_0^T & \cdots & \mathbf{b}_{N-1}^T \\ \mathbf{c}_0 & A_{00} & \cdots & A_{0N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{c}_{N-1} & A_{N-10} & \cdots & A_{N-1N-1} \end{pmatrix} \quad (6.2)$$

where $A_{jj'}$ are $k \times k$ matrices with entries $a_{jj'}(s)$, $s = 0 \dots k-1$. Clearly, these matrices are cyclic. \mathbf{b}_j denotes the vector $[b_j, \dots, b_j]^T$ of size k with equal entries; similarly, \mathbf{c}_j is the vector $[c_j, \dots, c_j]^T$.

A cyclic matrix can be reduced to a diagonal form using the DFT. Let $\mathcal{D} = \text{diag}(1, \frac{1}{k}D_k, \dots, \frac{1}{k}D_k)$ where D_k is the DFT matrix of size k . The number of DFT blocks in \mathcal{D} is N .

Applying the transform to S , we obtain

$$\mathcal{D}S\mathcal{D}^{-1} = \begin{pmatrix} a_{00} & \mathbf{b}_0^T \overline{D}_k & \cdots & \mathbf{b}_{N-1}^T \overline{D}_k \\ \frac{1}{k}D_k \mathbf{c}_0 & \frac{1}{k}D_k A_{00} \overline{D}_k & \cdots & \frac{1}{k}D_k A_{0N-1} \overline{D}_k \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{k}D_k \mathbf{c}_{N-1} & \frac{1}{k}D_k A_{N-10} \overline{D}_k & \cdots & \frac{1}{k}D_k A_{N-1N-1} \overline{D}_k \end{pmatrix}$$

The matrices $(1/k)D_k A \overline{D}_k$ are diagonal with entries on the diagonal $D_k \mathbf{a}_{jj'}$, where $\mathbf{a} = [a_{jj'}(0) \dots a_{jj'}(k-1)]$. Note that vectors $D_k \mathbf{b}_j$ and $D_k \mathbf{c}_n$ have zeros in all positions except the first: $D_k \mathbf{b}_j = [kb_j 0 \dots 0]^T$, $D_k \mathbf{c}_j = [c_j 0 \dots 0]^T$.

Finally, the subdivision matrix can be reduced to block diagonal form by applying a permutation. Let P be the permutation that rearranges the entries of a vector of length $kN + 1$ as follows: $[0, 1, 2, 3, \dots, Nk] \rightarrow [0, 1, k+1, \dots, (N-1)k+1, 2, k+2, \dots, (N-1)k+2, \dots, Nk]$ Applying this permutation we obtain

$$P\mathcal{D}S\mathcal{D}^{-1}P^{-1} = \text{diag}\left(Z, B\left(e^{\frac{2\pi i}{k}}\right), \dots, B\left(e^{\frac{2(k-1)\pi i}{k}}\right)\right) \quad (6.3)$$

The matrix has $k-1$ $N \times N$ blocks $B(\omega)$ where $\omega = e^{2\pi i/k}, \dots, e^{2(k-1)\pi i/k}$. Each $B(e^{2m\pi i/k})$ has entries $[D\mathbf{a}_{jj'}]_m$, i.e., is composed of m -th entries of DFT transforms of all vectors $\mathbf{b}_{jj'}$. For $m = 0$ we have to consider a larger $(N+1) \times (N+1)$ matrix Z with vectors $\mathbf{b} = [b_0, \dots, b_{N-1}]^T$ and $(1/k)\mathbf{c} = (1/k)[c_0, \dots, c_{N-1}]^T$ added on two sides. Note that $B(e^{2m\pi i/k}) = \overline{B(e^{2(k-m)\pi i/k})}$ and the eigenvalues of these blocks are conjugate. If an eigenvalue happens to

be real, and corresponds to the block $B(e^{2m\pi i/k})$ with $m \neq k/2$, it necessarily has an eigenspace of dimension at least 2. If x is its complex eigenvector obtained from an eigenvector of $B(e^{2m\pi i/k})$, a pair of real eigenvectors in this subspace can be taken to be $\Re x$ and $\Im x$. If an eigenvalue λ is complex, the two-dimensional real eigenspace corresponding to λ and $\bar{\lambda}$ is also spanned by $\Re x$ and $\Im x$.

Keeping in mind the support size of the scheme, it is easy to show that each block B also has a particular structure, for a suitable choice of numbering of vertices in each sector.

$$B(\omega) = \begin{pmatrix} B_{00}(\omega) & 0 \\ B_{10}(\omega) & B_{11}(\omega) \end{pmatrix}$$

In this way, the size of the matrices that have to be analyzed is further reduced; for example, for the Butterfly scheme considered in Section 7 $B(\omega)$ is 6×6 , and $B_{00}(\omega)$ is 3×3 .

Necessary condition for tangent plane continuity of invariant schemes. Before we proceed to the analysis of specific subdivision schemes, we formulate a necessary condition for C^1 -continuity. We need this condition to show that the Butterfly scheme is *not* C^1 -continuous for most valences.

Each eigenvalue of the subdivision matrix is an eigenvalue of a block $B(e^{2m\pi i/k})$, $m = 1 \dots k-1$ or Z . Each eigenvector can be obtained by taking an eigenvector of one of the blocks, setting the rest of the entries to 0, and transforming it using DP . This means that the eigenvectors have symmetries that can be used to establish necessary conditions on location of dominant eigenvalues of the subdivision matrix.

A condition of this type was proposed in [14] (Theorem 3.1). The theorem of Peters and Reif states that the dominant eigenvalues for a subdivision scheme with injective characteristic map necessarily have to be the eigenvalues of the blocks $B(e^{2\pi i/k})$ and $B(e^{2(k-1)\pi i/k})$. Intuitively, it appears that this is true for any ‘‘reasonable’’ subdivision scheme. However, it is possible to construct examples of C^1 -continuous schemes with eigenvalues corresponding to the characteristic map being in other blocks. Typically, such schemes would have noninjective characteristic map. Injectivity of a characteristic map is not strictly necessary for C^1 -continuity of the scheme, contrary to Theorem 2.2 of [14]. However, the cases when the scheme is C^1 -continuous and the characteristic map is not injective, are quite degenerate and are unlikely to be practically useful.

A weaker version of the conditions of Peters and Reif under some additional assumptions is proved in [19]. We a further simplified version of the condition, which is sufficient for our purposes.

As the subdivision matrix for an invariant scheme can be reduced to the block diagonal form, each cyclic subspace of the matrix is also a cyclic subspace of one of Z , $B(\omega)$, $\omega = e^{2\pi i/k}, \dots, e^{2(k-1)\pi i/k}$.

Lemma 6.1. *Suppose that the subdivision matrix for a subdivision scheme has a pair of dominant cyclic subspaces J_b^a , J_d^c , which either coincide and both have order 1, or are distinct and have order 0. Suppose these subspaces correspond to the blocks $B(e^{2\pi m i/k})$ and $B(e^{2\pi(k-m)i/k})$, $m \neq 1$, and the Jacobian of the characteristic map of this pair of cyclic subspaces is not identically zero. Let λ be an eigenvalue of the block $B(e^{2\pi i/k})$ and x a corresponding complex eigenvector.*

Suppose that for the limit map $f : U_1 \rightarrow \mathbf{R}^2$ generated by the pair $\Re x$, $\Im x$ the following two conditions hold: $f^{-1}(0) = \{0\}$, and the winding number of the curve obtained by restricting f to the boundary of U_1 is 1. Then the scheme is not C^1 -continuous.

Proof. see [19] for the proof. □

7 Analysis of the Butterfly and Modified Butterfly Schemes

The Butterfly subdivision scheme was proposed by Dyn, Gregory and Levin in [6]. In [7], it was proved that the scheme produces C^1 -continuous limit functions for regular meshes. Here we present an analysis of the scheme near extraordinary vertices. It turns out that for valences $k = 3$ and $k \geq 8$ the scheme is not C^1 -continuous. We also show that the Modified Butterfly scheme [21] is C^1 -continuous for all valences.

Definition of the schemes. The Butterfly scheme is an interpolating scheme: once a vertex is added to the complex, the control point corresponding to the vertex does not change. In [6], the coefficients of the scheme are parameterized by a parameter w . The scheme has maximal approximation order for $w = 1/16$; we analyze the scheme for this value

of w . The mask of the subdivision rule for newly inserted vertices is shown in Figure 4 on the left. The attractive feature of the scheme is its simplicity: the rules are the same for all vertices. However, as we will prove, the scheme does not produce C^1 -continuous surfaces.

In [21] we have proposed a modification of the Butterfly scheme, which does not have this problem. The rule for the immediate neighbors of an extraordinary vertex is modified in such a way that the spectrum of the subdivision matrix is similar to the spectrum of the subdivision matrix for valence 6, i.e. has eigenvalues 1, $1/4$ in block $B(0)$, $1/2$ in blocks $B(e^{2\pi i/k})$ and $B(e^{2(k-1)\pi i/k})$, and $1/2$ in blocks $B(e^{4\pi i/k})$ and $B(e^{2(k-2)\pi i/k})$. The rest of eigenvalues should be less than $1/8$. In order to achieve this, we use a mask with coefficients s_0, \dots, s_{k-1} , as shown in Figure 4 on the right. Note that this mask is asymmetric. For vertices on level finer than 0, this is not a problem: we are modifying coefficients of the scheme only for neighbors of extraordinary vertices, and only one of the two neighbors can be extraordinary after one subdivision step. On the top level both neighbors can be extraordinary. The choice that we make on the top level does not affect C^1 -continuity. We make an *ad hoc* choice to take the average of the results produced by each of the two possible choices. For $k \geq 5$ the coefficients are $s_j = (1/k)(1/4 + \cos(2\pi/k) + 1/2 \cos(4\pi/k))$, $j = 0, \dots, k-1$. For $k = 3$ we use $s_0 = 5/12, s_{1,2} = -1/12$, and for $k = 4, s_0 = 3/8, s_2 = -1/8, s_{1,3} = 0$. The properties of the scheme are discussed in greater detail in [21, 19].

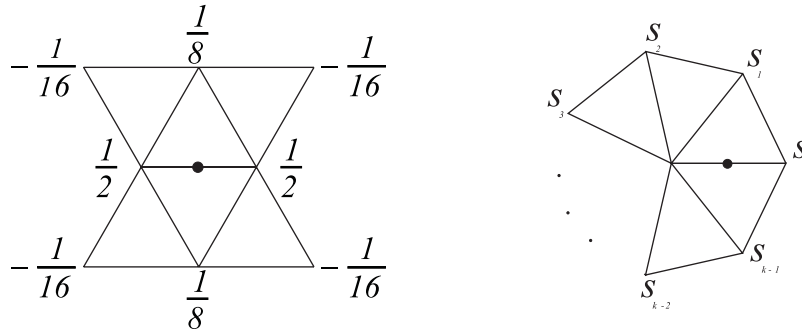


Figure 4: The masks of the Butterfly and Modified Butterfly schemes.

Subdivision matrices. For the Butterfly scheme, the size of the blocks $B(\omega)$ (Section 6) is 6×6 . There is no need to consider 0th block separately, as it can be split into a trivial 1×1 block and a 6×6 block.

All blocks have eigenvalues 0 and $-1/16$, the eigenvalue $-1/16$ having multiplicity 2 for each block. The other eigenvalues are eigenvalues of the upper left 3×3 subblock $B_{00}(\omega)$; these subblocks have the form

$$B_{00}(\omega) = \begin{pmatrix} \frac{1}{2} + \frac{1}{4}c - \frac{1}{8}(2c^2 - 1) & -\frac{1}{16}\bar{\omega} - \frac{1}{16} & 0 \\ \frac{1}{2} + \frac{1}{2}\omega - \frac{1}{16}\bar{\omega} - \frac{1}{16}\omega^2 & \frac{1}{8} & -\frac{1}{16} - \frac{1}{16}\omega \\ 1 & 0 & 0 \end{pmatrix}$$

where $c = \Re\omega = \cos(2m\pi/k)$. The characteristic polynomial of this matrix is

$$P(\lambda, d) = \lambda^3 + (-1/4 - 3/2d + d^2)\lambda^2 + \left(\frac{1}{64} + \frac{23}{64}d - 3/16d^2\right)\lambda - \frac{1}{64}d$$

where $d = c^2$.

For the Modified Butterfly scheme, there first row of $B_{00}(\omega)$ is replaced by $[\lambda(\omega), 0, 0]$, where $\lambda(\omega)$ is the prescribed largest eigenvalue for the block $B(\omega)$. The roots of the characteristic polynomial in this case are $\lambda(\omega)$, 0 and $1/8$.

Convergence rates. For both schemes, which coincide on the regular complexes, we use the contraction function $\|\nabla p\|_\infty$, with the difference operator $\nabla = [\Delta_{(1,0)}, \Delta_{(0,1)}] : (\ell_2^\infty) \rightarrow (\ell_2^\infty)^2$ (cf. [7]). For the difference scheme acting on the vectors $[\Delta_{(1,0)}p, \Delta_{(0,1)}p]$ we use the difference operator $\nabla' = [\Delta_{(0,1)}, \Delta_{(1,1)}, \Delta_{(1,0)}, \Delta_{(1,1)}] : (\ell_2^\infty)^2 \rightarrow (\ell_2^\infty)^4$. The convergence constants for the Butterfly scheme are $c = 1/2, \gamma_1 = 7/8, \gamma_2 = 31/64, \gamma_3 = 261/1024$. The constants for the difference scheme are $c^D = 1/2, \gamma_1^D = 1, \gamma_2^D = 7/8, \gamma_3^D = 11/16$.

We have chosen to use γ for 3 levels of subdivision, as after 3 levels of subdivision the convergence factor *per level* is close enough to what we would get if we were to use more levels: $\gamma_3^{1/3}$ is close to $\gamma_4^{1/4}$.

These estimates indicate that the convergence for derivatives is quite poor: γ stays close to 0.9 per level. However, this is a worst case estimate and in practice the scheme converges much faster. The reason for this is that for schemes with negative coefficients, the “worst case” happens when the initial values have changing signs, i.e., consist primarily out of high frequency components, which is uncommon for surfaces.

Analysis of the Butterfly scheme. The following proposition summarizes the information about the roots of the characteristic polynomial that we need to analyze the scheme. Whenever an approximate value of a root is given, it is implied that the precision is given by the last digit. Roots as functions of c are shown in Figure 5. The proof is straightforward, but tedious. An outline is presented in Appendix A.1. A detailed proof is contained in the Appendix B.

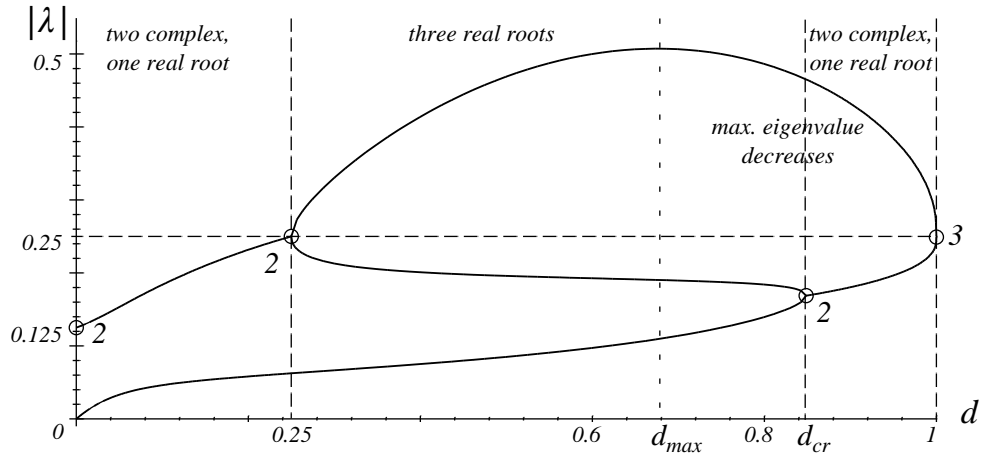


Figure 5: The magnitudes of the eigenvalues of the block $B_{00}(\omega)$ as functions of $d = \cos^2(\omega)$ for the Butterfly scheme. The vertical lines indicate the values of d for which the matrix has a nontrivial Jordan structure. The circles indicate eigenvalues with multiplicity greater than 1, the numbers next to the circles are multiplicities.

Proposition 7.1.

- For $d \in (1/4, d_{cr})$, $d_{cr} \approx 0.84868$. there are three real roots. The largest root is real, and is greater than $1/4$; the other roots are less than $1/4$.
- For $d \in (0, 1/4) \cup (d_{cr}, 1)$ there are two complex and one real root. The magnitude of the complex roots is always less than $1/4$, For $d \in (d_{cr}, 1)$ the real root is greater than $1/4$, for $d \in (0, 1/4)$, it is less than $1/4$.
- For $d = 0$, the characteristic polynomial has a double root $1/8$, and a single root 0 .
For $d = 1/4$, there is a double root $1/4$ and a single root $1/16$.
For $d = d_{cr}$, there is a single root $\lambda_1 \approx 0.46503$ and a double root $\lambda_2 \approx 0.16887$.
For $d = 1$, there is a triple root $1/4$.

For $d \in (d_{max}, 1)$, $d_{max} \approx 0.67600$, the largest root decreases as a function of d .

Using the information about the eigenvalues given by Proposition 7.1, we conclude that for $k \geq 10$, the maximum eigenvalue of the second block $B(4\pi/k)$, is greater than the largest eigenvalue of the first block $B(2\pi/k)$. Evaluating eigenvalues for $k = 3 \dots 9$ directly, we can see that this is also true for $k = 8$ and $k = 9$. For $k = 3$, ($d = 1/4$), the blocks $B(2\pi/3)$, and $B(4\pi/3)$, have double eigenvalues $1/4$.

Once the eigenvalues are known, the eigenvectors corresponding to the pair of dominant eigenvalues can be found from the complex eigenvector of $B(\omega)$ given by

$$e_0(\omega) = \left[\lambda, 1, \frac{(\omega + 1)((2c - 9)\lambda + 1)}{2 - 16\lambda} \right]$$

To compute the pair of eigenvectors defining the characteristic map, we first extend $e(\omega)$ to an eigenvector of a full 6×6 block using $v(\omega) = (\lambda I - B_{11})B_{10}e_0$. Then the complete complex eigenvector \mathbf{e} for valence k is $[0, e(0), e(2\pi/k), e(4\pi/k), \dots, e(2(k-1)\pi/k)]$. From this vector we obtain two real vectors $\Re \mathbf{e}$ and $\Im \mathbf{e}$, defining the characteristic map.

The algorithms that we use to check regularity of the characteristic map with minor changes can be used to verify the assumptions of Lemma 6.1. For valences $k = 4, 5, 7$ we use these algorithms to show injectivity and regularity of the characteristic map.

Our findings are summarized in the following proposition:

Proposition 7.2. *The Butterfly scheme is C^1 -continuous for valences $k = 4, 5, 6, 7$; it is not C^1 -continuous for any other valence, and is not tangent plane continuous for $k = 3$.*

While the scheme is not formally C^1 -continuous, the actual appearance of the surfaces generated by the scheme is not obviously non-smooth: for valences other than 3, the scheme produces tangent plane continuous surfaces, and the “twist” that makes the surfaces non- C^1 -continuous is a relatively subtle effect. For more details, see [19].

Modified Butterfly scheme. As the eigenvalues of the subdivision matrix in this case are prescribed, no eigenvalue analysis is necessary. The eigenvectors can be determined in the same way as it was done for the Butterfly scheme.

The control mesh for the ring consists of 6 rings of vertices around the central vertex as shown in Figure 6.

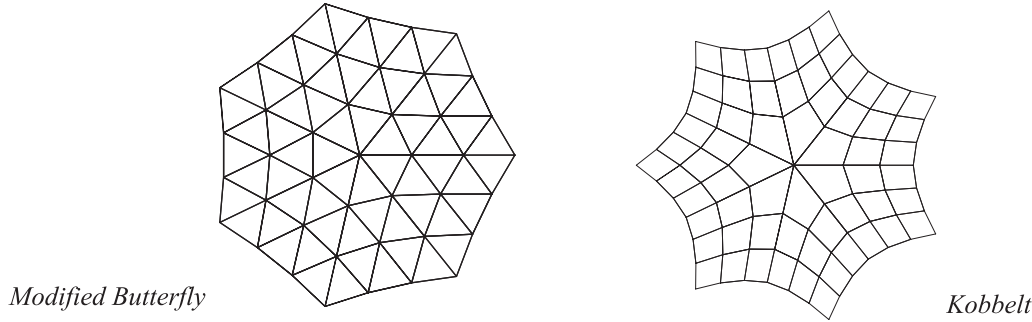


Figure 6: Control nets of the characteristic maps for the Modified Butterfly scheme and the Kobbelt scheme.

The convergence rates for the Modified Butterfly scheme are exactly the same as for the Butterfly scheme, as these schemes coincide on the regular complexes. We used our algorithms for verification of regularity and injectivity of the characteristic map to prove that the scheme is C^1 -continuous for any fixed valence.

As it was discussed above, it is possible to prove convergence for all valences if suitably chosen affine transforms of the control nets for one segment of the characteristic map converge to a limit as $k \rightarrow \infty$ and the normalized segment in the limit is regular and injective. This is the case for the Modified Butterfly scheme; the affine transform that we use is simply scaling along the y -axis by $\sin(2\pi/k)$.

Normalized control nets for several valences and the limit mesh are shown in Figure 7

The algorithm of Section 5 steps through the valences, verifying C^1 -continuity for each valence which has sufficiently different control net. In the case of the Modified Butterfly scheme we were able to use only a relatively small step size 2.6×10^{-6} , with all tests passing only after 6 steps of subdivision. once for a given scheme, time becomes a concern only for multiparameter schemes. The maximal valence k_0 that had to be tested (1481) is determined by the condition $|\cos(2\pi/k) - 1| < \delta$, such that the tests succeed for $\mathbf{e}(\lambda([1 - \delta, 1]))$ (recall that all quantities are represented as intervals). For each tested valence, we increase the interval size for control points, in order to be able to analyze many valences simultaneously for large valences. We have used the interval size 0.7×10^{-5} for valence greater than 6. The total number of valences that had to be analyzed separately was 406.

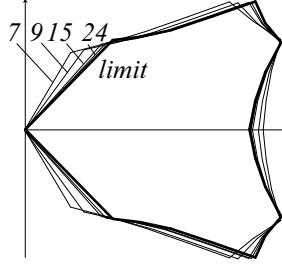


Figure 7: The convergence of the normalized control meshes of one segment of the characteristic maps for the Modified Butterfly scheme as valence increases. Only the boundaries of the meshes are shown.

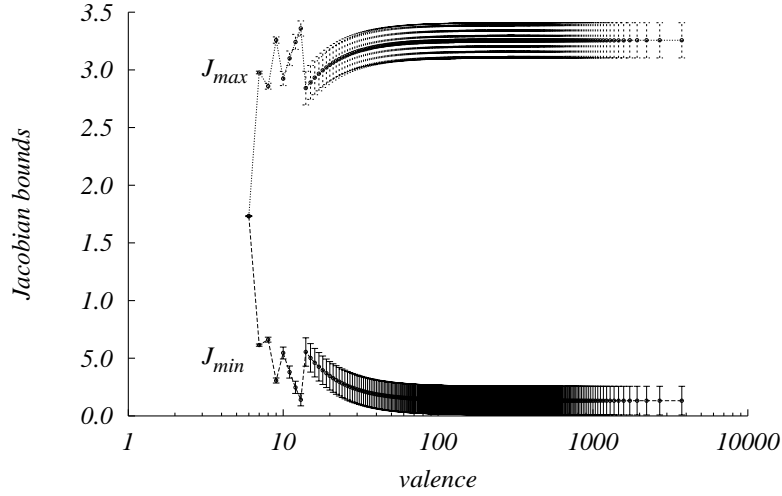


Figure 8: The upper and lower bounds for the Jacobians; the error bars show the interval for each bound; the step of the algorithm was chosen to be 2.6^{-6} so that the lower bound of the interval for J_{min} is close to zero. The initial anomaly in the lower bound is due to the fact that 8 subdivision steps, rather than 6 as for all other valences, were required to verify regularity for valence 3.

8 Analysis of Kobbelt Scheme

Kobbelt's subdivision scheme [12], is an interpolatory scheme defined on quad meshes; in the regular case, the scheme reduces to the tensor product of four-point schemes [5]. There are two challenges in the analysis of this scheme: first, as for the Butterfly scheme, the limit surface can not be expressed in explicit form. In addition, the eigenvalues of the subdivision matrix cannot be found explicitly.

Let $\alpha = (8 + w)/16$ and $\beta = -w/16$ be the coefficients of the four point scheme, where w is a parameter. Let $p_{i,l}^j$ be the control point corresponding to the vertex with number l in sector i at subdivision level j . Then the control points for level $j + 1$ are computed from the values on level j in two steps. First, the *edge points* are computed; all vertices are computed in the usual manner using the four-point rule, excluding the vertices $p_{i,1}^{j+1}$ immediately adjacent to the extraordinary vertex. These vertices are computed using the formulas

$$\begin{aligned}
 p_{i,1}^{j+1} &= \alpha c + \alpha p_{i,1}^j + \beta u_i^j + \beta p_{i,3}^j \\
 u_i^j &= \frac{4}{k} \sum_{i=0}^{k-1} p_{i,1}^j - (p_{i-1,1}^j + p_{i,1}^j + p_{i+1,1}^j) - \frac{\beta}{\alpha} (p_{i-2,2}^j + p_{i-1,2}^j + p_{i,2}^j + p_{i+1,2}^j) + \frac{4\beta}{\alpha k} \sum_{i=0}^{k-1} p_{i,2}^j
 \end{aligned} \tag{8.1}$$

where u_i^j are intermediate “virtual points”.

Next, the *face points* are computed. All face points are computed in the same way: four-point coefficients are applied to four consecutive edge points on level $j + 1$, as shown in Figure 9. It is important to note that there are two ways to choose four consecutive edge points; the coefficients for the scheme are chosen in such way that both choices produce the same result.

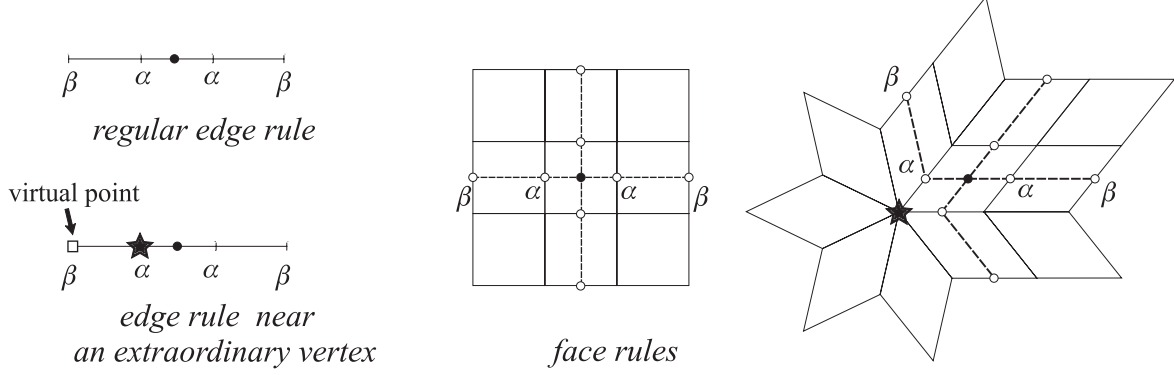


Figure 9: Rules for the Kobbelt scheme. The stars indicate extraordinary vertices, $\alpha = (8 + w)/16$, $\beta = -w/16$. In the masks for face control points (right) empty circles are edge vertices inserted on the *same* subdivision step. The dashed lines show the two possible sequences of four edge points that are used to compute a face point.

We performed the analysis of the scheme for $w = 1$, which is the value for which the four point scheme has maximal smoothness. After the standard operation of applying DFT to the subdivision matrix, we obtain the following 12×12 matrix $B(\omega)$:

$$\begin{bmatrix}
 c_{00} & c_{01} & \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 c_{10} & c_{11} & (1+\omega)\alpha\beta & \beta^2\omega+\alpha\beta & \beta^2 & \beta^2\bar{\omega}+\alpha\beta & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 \alpha & \alpha+\beta\bar{\omega} & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 \alpha\omega & \alpha+\beta\omega & 0 & \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 \hline
 \alpha & 0 & \alpha & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 \\
 \beta^2\bar{\omega}+\alpha^2+\alpha\beta\omega & \alpha\beta\bar{\omega}+\alpha^2 & \beta^2\omega+\alpha^2 & \alpha^2 & \alpha\beta & \alpha\beta(1+\bar{\omega}) & \alpha\beta & \alpha\beta & \beta^2 & 0 & 0 & \beta^2\bar{\omega} \\
 \beta\omega & \alpha & 0 & \alpha & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 \\
 (1+\omega)\alpha\beta & \alpha^2 & (1+\omega)\alpha\beta & \alpha^2 & \alpha^2 & \alpha^2 & \beta^2(1+\omega) & \alpha\beta & \alpha\beta & \beta^2 & \alpha\beta & \alpha\beta \\
 \beta & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & 0 & 0 & \beta \\
 \alpha\beta+\alpha^2\omega+\beta^2\omega^2 & \alpha^2+\alpha\beta\omega & \beta^2+\alpha^2\omega & (1+\omega)\alpha\beta & \alpha\beta & \alpha^2 & \alpha\beta\omega & \beta^2\omega & 0 & 0 & \beta^2 & \alpha\beta
 \end{bmatrix} \quad (8.2)$$

where $\omega = e^{2im\pi/k}$, $m = 1 \dots k$, k is the valence, and where

$$\begin{aligned}
 c_{00} &= \alpha + 4\beta\delta_{m,0} - \beta(1+2c) & c_{01} &= 4\frac{\beta^2}{\alpha}\delta_{m,0} - \frac{\beta^2}{\alpha}(\bar{\omega}^2 + 2c + 1) \\
 c_{10} &= 4\beta\alpha\delta_{m,0} + \alpha^2(1+\omega) - (1+\omega)\alpha\beta & c_{11} &= 4\beta^2\delta_{m,0} - \beta^2(1+2c) + 2\alpha\beta c + \alpha^2
 \end{aligned}$$

a

As it is discussed in 6, for any subdivision scheme each block can be separated into subblocks, with eigenvalues of lower-right 6×6 block not depending on the valence and equal to $-1/16$, $-1/32$, $-1/64$ (double), $-1/128$, and $1/256$.

The larger eigenvalues are always eigenvalues of the upper-left 6×6 block. The roots of the characteristic polynomial of that block cannot be found explicitly. However, for fixed m and k , we can easily find the roots numerically, with guaranteed lower and upper bounds on the roots. The characteristic polynomial has the form

$$P(c, \lambda) = \lambda^6 + \left(-\frac{3}{64}c - \frac{15}{16}\right)\lambda^5 + \left(\frac{297}{1024} - \frac{9}{1024}c\right)\lambda^4 + \left(-\frac{335}{8192} + \frac{21}{4096}c - \frac{7}{16384}c^2\right)\lambda^3 \quad (8.3)$$

$$+ \left(\frac{183}{65536} - \frac{9}{16384}c\right)\lambda^2 + \left(-\frac{45}{524288} + \frac{9}{524288}c\right)\lambda + \frac{1}{1048576} \quad (8.4)$$

Numerically computed roots of this polynomial are plotted as functions of c in Figure 10

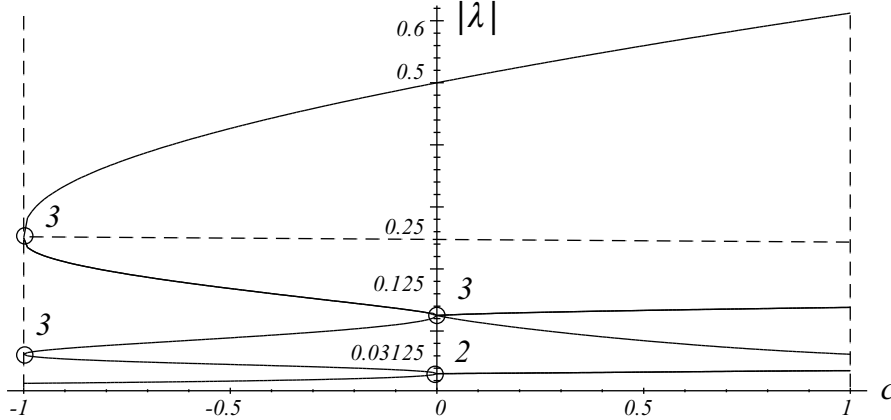


Figure 10: The magnitudes of the eigenvalues of the subdivision matrix for Kobbelt's scheme as functions of $c = \cos \frac{2m\pi}{k}$. Only the eigenvalues of the upper-right 6×6 block are shown. Note that the magnitudes of the complex conjugate eigenvalues coincide, and there are fewer than 6 distinct curves.

Analysis of the eigenvalues. From the plot it is clear that the largest eigenvalue increases as a function of c ; therefore, it appears that the largest eigenvalue of the subdivision matrix for any valence corresponds to $m = 1$. Moreover, our calculations indicate that the largest eigenvalue is always real. Using interval methods, we prove the following proposition:

Proposition 8.1. *For any valence k , and any $m = 1 \dots k - 1$ the largest eigenvalue is real and unique, and for any block $B(2m\pi/k)$, $m \neq k - 1, 1$ the largest eigenvalue is less than the largest eigenvalue of the blocks $B(2\pi/k)$ and $B(2\pi(k - 1)/k)$. The unique largest eigenvalue is the only eigenvalue in the interval $[0.5, 0.613]$, for $k > 4$.*

The detailed proof with all calculations, including the Maple code, be found in Appendix B.

Here we present an outline of the proof. The proof is performed in several steps:

1. We show that for $c < 0$, all roots of the characteristic polynomial $P(c, \lambda)$ are likely to be less than 0.51 (actually, they are less than 0.5, but due to numerical nature of our calculations, we have to relax the upper boundary).
2. We show that for any $c \in [0 \dots 1]$, there is a unique real root μ in the interval $[0.5, 0.613]$, and the function $\mu(c)$ is C^1 -continuous and increases.
3. We "deflate" the characteristic polynomial (that is, divide by the monomial $\lambda - \mu$), and verify that all roots of the deflated polynomial are inside the circle of radius 0.5 for $c \in [0, 1]$.

Marden-Jury test. On steps 1 and 3 we have to show that the roots of a polynomial are inside a circle of radius r in the complex plane. This task is similar to the task of establishing stability of a filter with the transfer function $1/a(z)$, where $a(z) = \sum_{i=0}^M a_i z^i$ is a polynomial. The filter is stable, if all roots of the polynomial $a(z)$ are inside the unit circle. A variety of tests exist for this condition; for our purposes, the algebraic Marden-Jury test is convenient [11]. With appropriate rescaling of the variable it can be used to prove that all roots of a polynomial are inside the circle of any

given radius r . As the test requires only a simple algebraic calculation on the coefficients of the polynomial, it can be easily performed for symbolic and interval coefficients.

To perform the test for a real polynomial $\sum_{m=0}^M \lambda^m$, a table is constructed. The first line is given by $r_i^0 = a_i$, $i = 0 \dots M$. The rest of the table is defined recursively:

$$r_j^{i+1} = \begin{vmatrix} r_0^i & r_{M-i-j}^i \\ r_{M-i}^i & r_j^i \end{vmatrix}, \quad j = 0 \dots M - i$$

Each row contains one element less than the previous row. Once the table is computed, the necessary and sufficient condition for stability is $r_0^1 < 0$, $r_0^i > 0$ for $i = 1 \dots M$.

A more detailed discussion of the proof of Proposition 8.1 can be found in Appendix A.2.

Analysis of C^1 -continuity. Using Proposition 8.1, we can easily compute the largest eigenvalue with guaranteed bounds for any k : this will be the unique real root in the interval $[0.5, 0.613]$. We compute roots up to a maximal valence k_0 . Once the eigenvalues are known, the eigenvectors defining the characteristic map are computed, and the tests described in Section 5 are applied to establish C^1 -continuity of the scheme for any fixed valence.

For this scheme we use the standard difference operator ∇ defined in Section 3; because this is a tensor product scheme, the matrix difference scheme is diagonal and can be decomposed into scalar schemes. The convergence constants are $c = 13/32$, $\gamma_1 = 25/32$, $\gamma_2 = 105/256$, $\gamma_3 = 425/2048$. The convergence constants for the difference scheme are $c^D = 31/64$, $\gamma_1^D = 5/4$, $\gamma_2^D = 15/16$, $\gamma_3^D = 5/8$.

To complete analysis of the scheme we need to describe the behavior of $\mu(c)$ at infinity. Specifically, to use our algorithm for verification of smoothness for all valences, for any interval value $c = [1 - \epsilon, 1]$ we need to estimate the corresponding interval value $\mu(c)$. As $\mu(c)$ changes slowly, linear approximation is sufficient for our purposes; the upper bound for the derivative $\mu'_c = 1/c_\mu$ can be easily computed. This allows us to compute the interval eigenvectors at infinity, and verify C^1 -continuity for all valences greater than k_0 .

The control mesh for the characteristic map of the Kobbelt scheme for valence 7 is shown in Figure 6. Figure 11 shows the dependence of the upper and lower estimates of computed Jacobians on the valence. Valences up to 812 had to be examined; because eigenvectors for large valences were sufficiently close, it was possible to perform analysis for a number of valences simultaneously; thus, only 193 valences had to be tested.

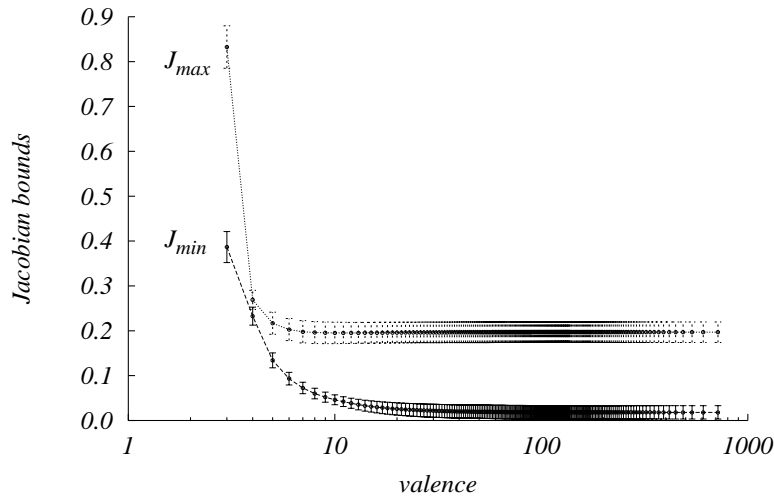


Figure 11: The upper and lower bounds for the Jacobian of the characteristic maps as functions of the valence for the Kobbelt scheme. The error bars indicate the size of the interval; the step of the algorithm was chosen to be 3×10^{-5} ; the maximal examined valence was 818; the total number of valences for which the test was performed was 193.

We conclude that *the Kobbelt scheme is C^1 -continuous for all valences.*

9 Other Schemes

C^1 -continuity of the Loop scheme was verified in [18] for valences up to 150. For the Catmull-Clark scheme, C^1 -continuity was analyzed for valences up to 10,000 in [14]. As the subdivision matrices have relatively simple form, and the eigenvalues and eigenvectors can be explicitly computed, our algorithms can be applied without any extra effort to obtain a proof of C^1 -continuity for all valences. Spline-based schemes have a remarkable property: the convergence rates γ_1 and γ_1^D for the scheme and the difference scheme are both $1/2$; this is due to the fact that both the scheme and the derivative scheme have only positive coefficients. In addition, replacing the coefficients of the scheme by intervals, we establish not only C^1 -continuity of the schemes, but also stability for small perturbations of the non-zero coefficients, as long as the perturbed coefficients lead to a convergent scheme. Due to fast convergence and high stability, intervals of large size can be used in the analysis, and only few valences (up to 58 for the Loop, up to 89 for Catmull-Clark) have to be analyzed. The number of subdivision iterations required to verify regularity is also quite small: 3 iterations are sufficient in both cases.

Our techniques can be also applied in virtually unchanged form to the dual, or corner cutting, subdivision schemes. Two schemes of this type are known to us: the Doo-Sabin subdivision scheme [4] and the Midedge subdivision scheme [9, 15]. For these schemes, C^1 -continuity was already proved for all valences [14, 9, 15]. Using our method, it is possible to perform perturbation analysis of the type that we have described above. We will discuss issues related to stability of smoothness properties of subdivision in greater detail in a future paper.

10 Conclusions

We have presented a general method for the verification of C^1 -continuity of stationary subdivision. This method allows us to analyze schemes which are not derived from spline subdivision, and perform most of the analysis automatically. Most of the difficulties in the analysis of the Butterfly and Kobbelt scheme can be eliminated, if the coefficients of the scheme are constructed taking the eigenstructure of the subdivision matrix into account. While this approach might lead to somewhat more complex expressions of coefficients, it is likely to have little if any effect on the performance, as the example of the Modified Butterfly scheme demonstrates. In the extreme case when the coefficients are computed from a set prescribed eigenvalues, the analysis of the eigenstructure becomes trivial.

Our method opens the way for a general characterization of invariant C^1 -continuous schemes with small support: such schemes are defined by a small number of parameters, and our interval algorithms can be used to prove C^1 -continuity for continuous ranges of these parameters.

Applications of our algorithms are not restricted to the invariant schemes for closed meshes: in fact, we have successfully used them to establish C^1 -continuity on the boundary of several variations of common subdivision schemes. These results are discussed in a future paper.

As it could be seen from our analysis of the Butterfly and Kobbelt subdivision schemes, the eigenstructure of a particular scheme or family of schemes still has to be analyzed manually. This stage requires determining the Jordan normal form of the subdivision matrix; in general, it is not always possible to do this numerically. However, we have demonstrated, using the Kobbelt scheme as an example, that it is possible to use semi-numerical methods to obtain all necessary information. We believe that a satisfactory solution of this problem requires methods similar to those developed in [8]. While it might not be possible to determine the Jordan normal form exactly, one can find all possible Jordan normal forms of matrices that are obtained from the original subdivision matrix by a small perturbation. This approach is likely to yield an algorithm that would be capable to perform the analysis C^1 -continuity of a scheme given only the coefficients of the scheme as the input.

While our method can be used to analyze parametric families of subdivision schemes, due to its semi-numeric nature, it cannot be used for such tasks as finding precise ranges of the values of the parameters for which the scheme remains C^1 -continuous. While this is of little relevance for practical applications, it can be regarded as a theoretical drawback.

References

- [1] A. A. Ball and D. J. T. Storry. Conditions for tangent plane continuity over recursively generated B-spline surfaces. *ACM Transactions on Graphics*, 7(2):83–102, 1988.

- [2] A. S. Cavaretta, W. Dahmen, and C. A. Micchelli. Stationary subdivision. *Memoirs Amer. Math. Soc.*, 93(453), 1991.
- [3] A. Cohen, N. Dyn, and D. Levin. Matrix subdivision schemes. Technical report, Université Pierre et Marie Curie, 1997.
- [4] D. Doo and M. Sabin. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, 1978.
- [5] N. Dyn, J. A. Gregory, and D. Levin. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4:257–268, 1987.
- [6] N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, April 1990.
- [7] N. Dyn, D. Levin, and C. A. Micchelli. Using parameters to increase smoothness of curves and surfaces generated by subdivision. *Computer Aided Geometric Design*, 7:129–140, 1990.
- [8] A. Edelman, E. Elmroth, and B. K. gström. A geometric approarturbation theory of matrices and matrix pencils. part i: Versal deformation. Technical report, MIT, 1996. preprint.
- [9] A. Habib and J. Warren. Edge and vertex insertion for a class of subdivision surfaces. Preprint. Computer Science, Rice University, 1996.
- [10] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Computer Graphics Proceedings*, Annual Conference Series, pages 295–302. ACM Siggraph, 1994.
- [11] E. I. Jury. *Theory and Applications of the z-Transform Method*. Wiley, New York, 1964.
- [12] L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Proceedings of Eurographics 96*, Computer Graphics Forum, pages 409–420, 1996.
- [13] R. E. Moore. *Methods And Applications Of Interval Analysis*. SIAM, Philadelphia, 1979.
- [14] J. Peters and U. Reif. Analysis of generalized B-spline subdivision algorithms. *SIAM Journal of Numerical Analysis*, 1997.
- [15] J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Transactions on Graphics*, 16(4), October 1997.
- [16] S. Pinchuk. A counterexample to the strong real Jacobian conjecture. *Mathematische Zeitschrift*, 217(1):1–4, 1994.
- [17] U. Reif. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design*, 12:153–174, 1995.
- [18] J. E. Schweitzer. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, Seattle, 1996.
- [19] D. Zorin. *Subdivision and Multiresolution Surface Representations*. PhD thesis, Caltech, Pasadena, 1997.
- [20] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes of arbitrary topology. Technical Report CS-TR-96-06, Caltech, Department of Computer Science, Caltech, 1996.
- [21] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. *Computer Graphics Proceedings (SIGGRAPH 96)*, pages 189–192, 1996.
- [22] D. N. Zorin. Smoothness of subdivision on irregular meshes. *Constructive Approximation*, 1998. submitted.

A Technical Proofs

In this appendix we outline the proofs of two propositions used in the analysis of the Butterfly and Kobbelt schemes. These proofs use symbolic and numeric computations. The complete Maple code with explanation is available separately.

A.1 Proposition 7.1

The roots of the characteristic polynomial of the Butterfly scheme can be found explicitly; depending on the value of $d = c^2$, there can be either 1 real and 2 complex roots, or 3 real roots. For four special values of d the matrix has nontrivial Jordan blocks; the special values of d are the roots of the discriminant of the characteristic polynomial, which is a polynomial in d . These roots are 0, $1/4$, 1, and $d_{cr} \approx 0.84868$. The types of roots depend on the sign of the discriminant of the characteristic polynomial; the discriminant is positive on $(0, 1/4)$ and $(d_{cr}, 1)$, negative on $(1/4, d_{cr})$. On each interval, well-known formulas can be used to find the roots of the characteristic polynomial as functions of d explicitly. To determine the largest root for any value of d , that is, the largest magnitude of an eigenvalue of a subblock $B_{00}(\omega)$ for a given ω , we consider the cases of 3 real roots and one real root separately.

Suppose $\lambda_i(d)$, $i = 1, 2, 3$ are the three real roots for $d \in [1/4, d_{cr}]$. As zero is a root only for $d = 0$, the three real roots do not change signs for $d \in [0, 1]$. It is sufficient to compute the value of roots at any point to show that all three roots are nonnegative. Therefore, the curves $|\lambda_1(d)|$, $|\lambda_2(d)|$, $|\lambda_3(d)|$ can intersect only if the roots coincide, which means that the discriminant is zero. Thus, the curves cannot intersect on the interval $(1/4, d_{cr})$. The largest root can be determined simply by evaluating the roots with guaranteed precision at any point of the interval.

If there is only one real root, we can easily show that for $d < 1/4$ $P(1/4, d) < 0$ and for $d > 1/4$ $P(1/4, d) > 0$. We conclude that for $d > d_{cr}$, the single real root is greater than $1/4$, and for $d < 1/4$ it is less than $1/4$. The magnitude of the complex roots also satisfies a cubic equation. Using the same method, we can show that these roots have magnitudes less than $1/4$ on $(1, 4)$ and $(3/4, 1)$.

Finally, we determine the range of d for which the largest root decreases as a function of d . Note that $\lambda_{max}(d)$ is a unique solution of the equation $P(\lambda, d)$ in the domain $(1/4, 1) \times (1/4, \infty)$. We can determine the zeros of the derivative $\lambda'_{max}(d)$ from the system of the equations $P(\lambda, d) = 0$, $P'_d(\lambda, d) = 0$, solving for d . Excluding λ using standard Gröbner basis techniques, and solving with guaranteed precision for d , we obtain the value d_{max} . As the derivative is not zero on $(d_{max}, 1)$, we determine the sign simply evaluating it at a point, and conclude that it is negative.

A.2 Proposition 8.1

Here we describe how steps 1-3 of the proof of Proposition 8.1 are performed.

Step 1. We have to verify that all roots of the polynomial for $c \in [0, 1]$ have magnitude less than 1. We split the interval into sufficiently small subintervals, so that we can evaluate Marden-Jury test in interval arithmetics for each subinterval with definite results.

The following observation is crucial for steps 2 and 3. Although the characteristic polynomial has degree 6 in λ , it is only quadratic in c , and has, two solutions, $c_1(\lambda)$ and $c_2(\lambda)$.

Step 2. Using interval evaluation of the derivative, one of the two solutions, say, $c_1(\lambda)$, can be shown to be increasing for $\lambda \in [0.5, 0.613]$. As $c_1(0.5) = 0$, and $c_1(0.613) > 1$, we conclude that for $c \in [0, 1]$, there is always a real solution in the range $[0.5, 0.613]$. If we evaluate the second root $c_2(\lambda)$ for the same interval of λ , we can observe that the values of c_2 are outside the range $[-1, 1]$. Therefore, for $c \in [0, 1]$ there is a unique real solution λ in the range $[0.5, 0.613]$. Because $c_1(\lambda)$ is C^1 -continuous and its derivative is positive, the inverse function $\mu(c)$ has the same properties (we use μ to distinguish between the real eigenvalue that we have identified from the indeterminate λ of the characteristic polynomial).

Step 3. To show that all other roots of the characteristic polynomial for $c \in [0, 1]$ are smaller than $\mu(c)$, we perform deflation symbolically, using μ as a parameter: we divide $P(c, \lambda)$ by $(\lambda - \mu)$ symbolically, and substitute $c = c_1(\mu)$. The coefficients of the resulting polynomial are functions of μ . Again, we separate the range of μ into subintervals, small enough to be able to obtain a definite result from the Marden-Jury test. This proves that for all values of μ in $[0.5, 0.613]$, and, therefore, for all $c \in [0, 1]$ $\mu(c)$ is the largest root of $P(\lambda, c)$.

The proposition is derived from the three statements in Section 8 in the following way.

As for $k > 4$, $\cos \frac{2\pi}{k} > 0.51$, the largest eigenvalue cannot possibly correspond to a block m , for which $\cos \frac{2m\pi}{k} \leq 0$. From step 3, it follows that the largest root has to be the real root $\mu(c)$ for some c . As for any $m > 1, m < k - 1$, $\cos \frac{2m\pi}{k} < \cos \frac{2\pi}{k}$, and we have shown on step 1 that $\mu(c)$, increases, and for any c $\mu(c)$ is the largest root (step 3), we conclude that the largest eigenvalue always corresponds to $m = 1$, is real, and is the unique eigenvalue in the range $[0.5, 0.613]$.

B Source Code

In this appendix, we have collected the Maple worksheets that contain the code for computing the eigenvalues and eigenvectors of several subdivision schemes: Butterfly, Modified Butterfly, Kobbelt, Loop and Catmull-Clark. These worksheets also contain the code used in the proofs of the facts about the eigenvalues that was used to analyze C^1 -continuity of these schemes. In addition, we include the worksheet with the code computing the convergence constants c and γ (Section 3).

We do not include the C++ code that implements the algorithms of Section 5, and the part of the Maple code that was used to generate C code. Complete sources are available from the author.

The worksheets are arranged in the following order:

- Convergence rates of matrix subdivision schemes.
- Eigenstructure of the Butterfly scheme.
- Eigenstructure of the Kobbelt scheme.
- Eigenstructure of the Loop scheme.
- Eigenstructure of the Catmull-Clark scheme.

Convergence Rates of Matrix Subdivision Schemes

Denis Zorin, 1997-98

Introduction

The procedures defined in this worksheet can be used to estimate the convergence rates of scalar and matrix subdivision schemes. It is based on the generalization to the matrix case of several theorems in "Stationary Subdivision" by Cavaretta, Dahmen and Micchelli [CDM], and to the multivariate case of several theorems from "Matrix subdivision" by Cohen, Dyn and Levin [CDL]. The goal is to compute four constants γ, c, γ_D, c_D , such that a given (possibly matrix) subdivision scheme S and its difference scheme S_D satisfy

$$D(S^n p) < \gamma D(p), D_1(S_D^n p) < \gamma_D D_1(p)$$

for some n and n_D , and a particular choice of contraction functions $D(p)$ and $D_1(p)$, and

$$|Sp - Bp| < cD(p), |S_D p - B_D p| < c_D D(p)$$

where B is the comparison scheme, B_D is the comparison scheme for the difference scheme.

We use commutation formulas to compute the constants (see the paper for details).

If a matrix scheme has a stable scaling function, it is C1 if and only if the difference scheme is C0 ([CDM, Th. 8.1,8.2], [CDL, Th. 5.2, 5.4] combined). Therefore, we can use second difference schemes in most cases to analyze C1-continuity.

We assume only that all schemes have constants as invariant space C (constant vectors for matrix schemes), that is, $SC = C$, which is a necessary condition for convergence to nonzero functions in the scalar case [CDM, Th. 2.3]. For the matrix schemes, it is not strictly necessary: it is only necessary that SC is a subset of C ; however, if it is a proper subset, it means that the components of the limit function are dependent [CDL, Prop. 2,1]. Our matrix schemes are primarily obtained as difference schemes, and all interesting cases are nondegenerate, i.e. $SC = C$. Whenever this is the case, the code will compute an estimate. However, if the estimate for the convergence rate is greater than 1, this does not necessarily mean that the scheme diverges. There may be two reasons for this. First, we estimate the convergence rate using the sup norm of the difference scheme of A^n , for user-specified n . For slowly converging schemes with some of the coefficients being negative, one may need arbitrarily large n , for which the norm cannot be practically computed. The second reason is more fundamental. Let D be the subspace of the space of sequence on which a matrix scheme S acts, such that any sequence in D can be represented as a vector of differences of values of some sequence p . For sufficiently large n , the infinity norm of a difference scheme restricted to D , is less than 1; however, we do not have a practical way of computing norms on D , and we compute the norm on the whole space; it is not necessary for this norm to be less than 1 for any n . In some cases (most notably, for the Butterfly scheme) these problems can be alleviated by choosing a suitable contraction function; our implementation allows one to choose contraction functions from a particular class (see the definitions of **ConvergenceEstimate** and **DiffScheme**). Moreover, somewhat better estimates can be obtained using a calculation specific to this type of schemes implemented in **FactorizableConvergenceEstimate**.

The main functions are **ConvergenceEstimate** and **FactorizableConvergenceEstimate**. The names **z1** and **z2**, as well as **gamma**, **c**, **gammaD**, **cD**, are global. Calculations for specific schemes are demonstrated in the last section. For the schemes listed there **FactorizableConvergenceEstimate** would be sufficient; however, in the general case we need **ConvergenceEstimate**, which will be used for non-factorizable schemes elsewhere.

Utilities

```
> with(linalg); kernelopts(ASSERT = true)
Warning, 'new definition for norm'
Warning, 'new definition for trace'
```

IsLaurentPolynom(A) Check if the argument is a scalar Laurent polynomial in $z1, z2$ with arbitrary coefficients.

```
> IsLaurentPolynom :=
```

```
proc(A::ratpoly(anything, [z1, z2])) global z1, z2; type(denom(factor(A)), monomial(anything, [z1, z2])) end
```

IsLaurentMatrix(A) Check if the argument is a matrix Laurent polynomial in $z1, z2$ with arbitrary coefficients.

```
> IsLaurentMatrix := proc(A::matrix(ratpoly(anything, [z1, z2])))
```

```
local res, ind;
```

```
res := true; for ind in indices(A) do res := res and IsLaurentPolynom(A[op(ind)]) od; res
```

```
end
```


▣ **IsIntegerVector2List(W)** Check if the argument is a list of vectors with integer components, each vector of length 2 each component in the range 0..3, and if one of the components is 0, the other is also 0. Used to check validity of arguments representing contraction functions.

```
> IsIntegerVector2List := proc(W::list(list(integer)))
  local i, res;
  res := true;
  for i to vectdim(W) do
    res := res and evalb(vectdim(op(i, W)) = 2);
    res := res and evalb(0 ≤ W[i][1] and W[i][1] ≤ 3);
    res := res and evalb(0 ≤ W[i][2] and W[i][2] ≤ 3);
    res := res and evalb(W[i][1] ≠ W[i][2]) or W[i][1] = 0 and W[i][2] = 0
  od;
  res
end
```

▣ **Type definitions**

```
> type/LaurentPolynom := eval(IsLaurentPolynom)
> type/LaurentMatrix := eval(IsLaurentMatrix)
> type/IntegerVector2List := eval(IsIntegerVector2List)
```

▣ **MakeContractionMatrix(W)** Make a matrix Laurent polynomial out of a list of vectors W such that **IsIntegerVector2List(W,1,3)** is true. The matrix is defined as $\text{diag}(p_1 \dots p_n)$, where p_i are vectors corresponding to vectors of W, obtained by replacing 1 with $z_1^2 - 1$, 2 with $z_2^2 - 1$ and 3 with $z_1^2 z_2^2 - 1$.

```
> MakeContractionMatrix := proc(W::IntegerVector2List)
  local p, i, Wmatrix;
  global z1, z2;
  p := [z1 - 1, z2 - 1, z1*z2 - 1];
  Wmatrix := matrix(2*vectdim(W), vectdim(W), 0);
  for i to vectdim(W) do Wmatrix[2*i - 1, i] := p[W[i][1]]; Wmatrix[2*i, i] := p[W[i][2]] od;
  evalm(Wmatrix)
end
```

▣ **MakePolynom(coef)** make a polynomial in z1,z2 out of a two-dimensional array of coefficients.

```
> MakePolynom := proc(polycoef::array(2))
  local ind, p;
  global z1, z2;
  p := 0; for ind in indices(polycoef) do p := p + z1^op(1, ind)*z2^op(2, ind)*polycoef[op(ind)] od; eval(p)
end
```

▣ **CoefList(A)** Make a 2d table of matrix coefficients of a matrix Laurent polynomial in 2 variables.

```
convert polynomials to tables of coefficient indexed by powers
> extractCoeffs := proc(x::LaurentPolynom)
  local i, dummy, polycoeffs, terms, powers, extrPowers;
  global z1, z2;
  polycoeffs := [coeffs(expand(factor(x)), [z1, z2], 'terms')];
  extrPowers := unapply(['degree'(dummy, z1), 'degree'(dummy, z2)], dummy);
  powers := map(extrPowers, [terms]);
  table([seq(op(op(i, powers)) = op(i, polycoeffs), i = 1 .. vectdim(polycoeffs))])
end
```

given a coefficient, make a matrix

```
> makeCoeffMatrix := proc(A::LaurentMatrix, i::integer, j::integer)
  local l, m, matrcoeff;
  matrcoeff := matrix(rowdim(A), coldim(A), 0);
  for l to rowdim(A) do
    for m to coldim(A) do if member([i, j], {indices(A[l, m])}) then matrcoeff[l, m] := A[l, m][i, j] fi od;
  od;
  eval(matrcoeff)
```

```

end
> CoeffList := proc(A::LaurentMatrix)
local x, matrcoeff, n, zeroMatrix, ld1, ld2, ud1, ud2, l, m, pcoeff, allpowers;
global extractCoeffs, makeCoeffMatrix;
pcoeff := map(extractCoeffs, A);
allpowers := map(unapply({`indices`(x)}, x), pcoeff);
allpowers := [op(`union`(op(map(op, [entries(allpowers)]))))]);
ld1 := min(op(map(unapply(`op`(1, x), x), allpowers)));
ld2 := min(op(map(unapply(`op`(2, x), x), allpowers)));
ud1 := max(op(map(unapply(`op`(1, x), x), allpowers)));
ud2 := max(op(map(unapply(`op`(2, x), x), allpowers)));
matrcoeff := array(ld1 .. ud1, ld2 .. ud2);
for n in allpowers do matrcoeff[op(n)] := makeCoeffMatrix(pcoeff, op(n)) od;
zeroMatrix := matrix(rowdim(A), coldim(A), 0);
for l from ld1 to ud1 do
    for m from ld2 to ud2 do if not member([l, m], allpowers) then matrcoeff[l, m] := evalm(zeroMatrix) fi od;
end
eval(matrcoeff)
end

```

GenerateCode(OutputFile, SchemeName, Consts, Mask, ControlSize, triangular) Generate code for a function that creates an instance of class RegScheme, defining a scheme. See file regscheme.h for class definition. **Consts** is a list of tables of convergence constants, each table generated by a call to **ConvergenceEstimates** or a similar function. **Mask** is the standard coefficient mask of the scheme.

ControlSize is the number of layers of control vertices outside a patch required to define the surface on the patch (clearly this number can be computed from the mask; however, computing the minimal number in the general case requires some care; typically, in each specific case this number is obvious, and we simply specify it explicitly); **triangular** indicates the symmetry type: if it is true, then 3-directional symmetry is assumed; otherwise 2-directional symmetry is assumed.

```

> GenerateCode := proc(
OutputFile::string, SchemeName::string, Consts::list(table), Mask::array(2), ControlSize::integer, triangular::boolean)
local i, g, dg, mask, mmin1, mmin2, mmax1, mmax2;
global S;
assert(0 ≤ ControlSize, `ControlSize must be nonnegative`);
fprintf(OutputFile, `RegScheme* %s(\n`, SchemeName);
mask := eval(Mask);
for i in indices(Mask) do if Mask[op(i)] = 0 then mask[op(i)] := BV else mask[op(i)] := Mask[op(i)] fi od;
mask := map(ConvertToFloat, eval(mask));
mmin1 := min(op(map(unapply(`op`(1, x), x), [indices(Mask)])));
mmin2 := min(op(map(unapply(`op`(2, x), x), [indices(Mask)])));
mmax1 := max(op(map(unapply(`op`(1, x), x), [indices(Mask)])));
mmax2 := max(op(map(unapply(`op`(2, x), x), [indices(Mask)])));
fprintf(OutputFile, `Float* g = new Float[%d];\n`, vectdim(Consts));
fprintf(OutputFile, `Float* dg = new Float[%d];\n`, vectdim(Consts));
fprintf(OutputFile, `Float** mask = new pFloat[%d];\n`, mmax1 - mmin1 + 1);
fprintf(OutputFile, `for( int i = 0; i < %d; i++)\n mask[i] = new Float[%d];\n`, mmax1 - mmin1 + 1,
mmax2 - mmin2 + 1);
g := array(1 .. vectdim(Consts));
dg := array(1 .. vectdim(Consts));
for i to vectdim(Consts) do g[i] := Consts[i][γ]; dg[i] := Consts[i][gammaD] od;
g := map(ConvertToFloat, eval(g));
dg := map(ConvertToFloat, eval(dg));
S := [ ];
C(mask);
C(g);
C(dg);

```

```

for i to vectdim(S) do fprintf(OutputFile, '%s', S[i]) od;
fprintf(OutputFile, 'return new RegScheme( %A, %d, g, %A, dg, new Grid<Float,Float>( %d, %d, %d, %d, mask), %d\
, %s); \n)\n\n', ConvertToFloat( Consts[ 1 ][ c ], vectdim( Consts ), ConvertToFloat( Consts[ 1 ][ cD ], mmin1,
mmax1, mmin2, mmax2, ControlSize, triangular );
NULL
end

```

Convergence estimates for bivariate matrix schemes

Note: all polynomials are required to be matrix polynomials; to use the functions for scalar polynomials, convert it to a 1 by 1 matrix first.

Decompose a bivariate matrix Laurent polynomial

IdealDecompose(A, i, j, symmetrize) Given a Laurent polynomial $A(z_1, z_2)$ known to be in the ideal generated by $z_1^2 - 1$ and $z_2^2 - 1$ find polynomials TA_1 and TA_2 such that $A = (z_1^2 - 1) TA_1 + (z_2^2 - 1) TA_2$, or $A = (z_1^2 z_2^2 - 1) TA_1 + (z_2^2 - 1) TA_2$ or $A = (z_1^2 - 1) TA_1 + (z_1^2 z_2^2 - 1) TA_2$; the expressions are derived from the expressions of Lemma 2.3 [CDM]. The decomposition is not unique; the expressions are asymmetric even in the first case; the choice of the pair of polynomials is given by the arguments i and j ; i corresponds to $z_1^2 - 1$, 2 to $z_2^2 - 1$ and 3 to $z_1^2 z_2^2 - 1$. The order matters; the last argument is used is used to request symmetrized decomposition with respect to $z_1^2 - 1$, and $z_2^2 - 1$; if one of i, j is 3, the last argument is ignored. The function returns a list of 2 polynomials **s**.

```

> IdealDecompose := proc(A::LaurentPolynom, i::integer, j::integer, symmetrize::boolean)
local TA1, TA2, Q, p;
global z1, z2;
ASSERT(1 ≤ i and i ≤ 3 and 1 ≤ j and j ≤ 3 and i ≠ j,
'second and third argument cannot coincide and should be in the range 1..3');
ASSERT(simplify(subs({ z1 = -1, z2 = -1 }, A)) = 0 and simplify(subs({ z1 = 1, z2 = -1 }, A)) = 0 and
simplify(subs({ z1 = -1, z2 = 1 }, A)) = 0 and simplify(subs({ z1 = 1, z2 = 1 }, A)) = 0,
'Laurent polynomial does not have all roots [(-1)^e, (-1)^g], e, g = 0, 1');
p[ 1 ] := z1^2 - 1;
p[ 2 ] := z2^2 - 1;
p[ 3 ] := z1^2*z2^2 - 1;
Q[ 1, 2 ] := (1 - z1)*subs(z1 = -1, A) + (1 + z1)*subs(z1 = 1, A);
Q[ 1, 3 ] := (1 - z1)*subs({ z2 = -z1*z2, z1 = -1 }, A) + (1 + z1)*subs({ z2 = z1*z2, z1 = 1 }, A);
Q[ 2, 1 ] := (1 - z2)*subs(z2 = -1, A) + (1 + z2)*subs(z2 = 1, A);
Q[ 2, 3 ] := (1 - z2)*subs({ z1 = -z1*z2, z2 = -1 }, A) + (1 + z2)*subs({ z1 = z1*z2, z2 = 1 }, A);
Q[ 3, 1 ] := (1 - z1*z2)*subs(z2 = -1 / z1, A) + (1 + z1*z2)*subs(z2 = 1 / z1, A);
Q[ 3, 2 ] := (1 - z1*z2)*subs(z1 = -1 / z2, A) + (1 + z1*z2)*subs(z1 = 1 / z2, A);
if symmetrize and i = 1 and j = 2 then
TA1 := factor(1*(2*A + Q[ 2, 1 ] - Q[ 1, 2 ]) / 4*(z1^2 - 1));
TA2 := factor(1*(2*A + Q[ 1, 2 ] - Q[ 2, 1 ]) / 4*(z2^2 - 1));
elif symmetrize and i = 2 and j = 1 then
TA2 := factor(1*(2*A + Q[ 2, 1 ] - Q[ 1, 2 ]) / 4*(z1^2 - 1));
TA1 := factor(1*(2*A + Q[ 1, 2 ] - Q[ 2, 1 ]) / 4*(z2^2 - 1));
else TA1 := factor(Q[ j, i ] / 2*p[ i ]); TA2 := factor((2*A - Q[ j, i ]) / 2*p[ j ])
fi;
if simplify(p[ i ]*TA1 + p[ j ]*TA2 - A) ≠ 0 then print('decomposition error') fi;
[ TA1, TA2 ]
end

```

IdealDecomposeMatrix(A, W, symmetrize) Given a matrix Laurent polynomial $A(z_1, z_2)$ known to be in the ideal generated by $z_1^2 - 1$ and $z_2^2 - 1$ find a matrix Laurent polynomial Q such that $A = QW(z_1^2, z_2^2)$; componentwise,

$$A_{i,j}(z_1, z_2) = Q_{i, 2j-1}(z_1, z_2) W_{j,1}(z_1^2, z_2^2) + Q_{i, 2j}(z_1, z_2) W_{i,2}(z_1^2, z_2^2)$$

Q is twice as wide as A and has the same number of rows. W is a list of pairs of integers p_i of length n equal to the width of A. Each integer is one of 1,2,3 corresponding to $z_1 - 1$, $z_2 - 1$ and $z_1 z_2 - 1$ respectively. In each pair, the integers cannot coincide. The matrix Laurent polynomial W is defined as $\text{diag}(p_1 \dots p_n)$. The last argument is used for the same purpose as in **IdealDecompose**.

```

> IdealDecomposeMatrix := proc(A::LaurentMatrix, W::IntegerVector2List, symmetrize::boolean)
  local Q, i, j, Wmatrix, testA, Qdecomp;
  global z1, z2;
  ASSERT(vectdim(W) = coldim(A),
    'the length of the 2nd argument (list) should be the same as width of the first argument (matrix)');
  Q := matrix(rowdim(A), 2*coldim(A));
  for i to rowdim(A) do for j to coldim(A) do
    if i mod 2 = 1 then
      Qdecomp := IdealDecompose(A[i, j], W[j][2], W[j][1], symmetrize);
      Q[i, 2*j - 1] := op(2, Qdecomp);
      Q[i, 2*j] := op(1, Qdecomp)
    else
      Qdecomp := IdealDecompose(A[i, j], W[j][1], W[j][2], symmetrize);
      ASSERT(IsLaurentPolynom(op(1, Qdecomp)) and IsLaurentPolynom(op(2, Qdecomp)),
        'decomposition failed');
      Q[i, 2*j - 1] := op(1, Qdecomp);
      Q[i, 2*j] := op(2, Qdecomp)
    fi
  od
od;
Wmatrix := MakeContractionMatrix(W);
testA := evalm(A - (Q &* (subs({ z1 = z1^2, z2 = z2^2 }, evalm(Wmatrix))));
ASSERT(norm(testA) = 0, 'scheme was not decomposed correctly');
evalm(Q)
end

```

☐ Compute a difference scheme for a matrix scheme

☐ **DiffScheme(A, W)** Compute the difference matrix scheme Q for a bivariate matrix scheme; the choice of directions for differences is given by W, defined as in **IdealDecomposeMatrix**. We assume that scheme is specified by its matrix Laurent polynomial $A(z_1, z_2)$, and reproduces constant vectors; the calculation solves the equation

$$W(z_1, z_2) A(z_1, z_2) = Q(z_1, z_2) W(z_1^2, z_2^2) \text{ componentwise,}$$

$$W_{i,1}(z_1, z_2) A_{i,j}(z_1, z_2) = Q_{2i-1,2j-1}(z_1, z_2) W_{j,1}(z_1^2, z_2^2) + Q_{2i-1,2j}(z_1, z_2) W_{i,2}(z_1^2, z_2^2)$$

$$W_{i,2}(z_1, z_2) A_{i,j}(z_1, z_2) = Q_{2i,2j-1}(z_1, z_2) W_{i,1}(z_1^2, z_2^2) + Q_{2i,2j}(z_1, z_2) W_{i,2}(z_1^2, z_2^2)$$

```

> DiffScheme := proc(A::LaurentMatrix, W::IntegerVector2List)
  local WA, Wmatrix, Q, testA;
  global z1, z2;
  ASSERT(coldim(A) = rowdim(A), '1st argument should be a square matrix');
  ASSERT(vectdim(W) = coldim(A),
    'the length of the 2nd argument (list) should be the same as either dimension of the 1st argument (matrix)');
  Wmatrix := MakeContractionMatrix(W);
  WA := evalm(Wmatrix &* A);
  Q := IdealDecomposeMatrix(WA, W, false);
  testA := evalm(WA - (Q &* (subs({ z1 = z1^2, z2 = z2^2 }, evalm(Wmatrix))));
  ASSERT(norm(testA) = 0, 'difference scheme was not found correctly');
  evalm(Q)
end

```

☐ Infinity norm for matrix polynomials

▣ **InfNnormSums(A, n)** compute the list of 4^n vectors of sums of magnitudes of entries of matrix coefficients of $A(z) A(z^2) A(z^4) \dots A(z^{2^n})$ where $z = (z_1, z_2)$; each component in each vector corresponds to a row in the matrix coefficients sums[i,j] includes all matrix coefficients of monomials $z^l z^k$ such that $l \bmod 2^n = i - 1$ and $k \bmod 2^n = j - 1$

```
> InfNnormSums := proc(A::LaurentMatrix, n::integer)
local x, pn, cfs, i, j, k, l, m, coeffs, ld1, ud1, ld2, ud2, ind1, ind2;
global z1, z2;
ASSERT(n = 1 or rowdim(A) = coldim(A), 'the matrix should be square if n > 1');
pn := A;
for i to n - 1 do pn := evalm(pn &* (subs( { z1 = z1^(2^i), z2 = z2^(2^i) }, evalm(A)))) od;
pn := map( expand, evalm(pn));
cfs := CoeffList(pn);
coeffs := array(0 .. 2^n - 1, 0 .. 2^n - 1);
for i from 0 to 2^n - 1 do for j from 0 to 2^n - 1 do coeffs[i, j] := vector(rowdim(A), 0) od od;
ind1 := map(unapply('op'(1, x), x), [indices(cfs)]);
ind2 := map(unapply('op'(2, x), x), [indices(cfs)]);
ld1 := min(op(ind1));
ud1 := max(op(ind1));
ld2 := min(op(ind2));
ud2 := max(op(ind2));
for l from ld1 to ud1 do for k from ld2 to ud2 do for m to coldim(A) do coeffs[l mod 2^n, k mod 2^n] :=
evalm( coeffs[l mod 2^n, k mod 2^n] + map(abs, col(cfs[l, k], m)))
od
od
od;
eval(coeffs)
end
```

▣ **InfNnormV(A, n)** get the vector of componentwise inf norms of $A(z) A(z^2) A(z^4) \dots A(z^{2^n})$ where $z = (z_1, z_2)$, as the componentwise maximum of the vector sums computed by InfNnormSums

```
> InfNnormV := proc(A::LaurentMatrix, n::integer)
local x, i, j, m, sums, vsumlist;
for i to rowdim(A) do vsumlist[i] := [ ] od;
sums := InfNnormSums(A, n);
for i from 0 to 2^n - 1 do
for j from 0 to 2^n - 1 do for m to rowdim(A) do vsumlist[m] := [ op(vsumlist[m]), sums[i, j][m] ] od od
od;
map(unapply('max'(op(x), x), vsumlist)
end
```

▣ **InfNnorm(A, n)** get the inf norm of $A(z) A(z^2) A(z^4) \dots A(z^{2^n})$ where $z = (z_1, z_2)$, as the maximum of all components of vector sums computed by InfNnormSums.

```
> InfNnorm := proc(A::LaurentMatrix, n::integer) max(op(map(op, [entries(InfNnormV(A, n))])) end
```

▣ Convergence rate estimates for a matrix scheme and its difference scheme

▣ **ConvergenceEstimates(A, n, nD, W, WD, B)** Compute convergence constants γ, c, γ_D, c_D , described in the introduction. The function returns a table indexed by names. The argument W allows one to define a contraction function $|W|_\infty$, similarly, WD defines the contraction function for the difference scheme. both W and WD are represented by lists of pairs of integers, as for **DiffScheme**. The comparison scheme for the difference scheme is taken to be the difference

scheme of B. This is not necessary in general, but convenient for our purposes. We assume that B has factors $\frac{w(z_1^2, z_2^2)}{w(z_1, z_2)}$,

where w is any of the polynomials $z_1 - 1$, $z_2 - 1$, $z_1 z_2 - 1$ used in W .

```

> ConvergenceEstimates := proc(
  A::LaurentMatrix, n::posint, nD::posint, W::IntegerVector2List, WD::IntegerVector2List, B::LaurentPolynom)
  local BD, Q, Q2, DA, DQ, BM, i, p, res;
  global z1, z2, c, cD, gammaD;
  ASSERT(rowdim(A) = coldim(A), '1st argument should be a square matrix');
  ASSERT(vectdim(W) = coldim(A),
    'the length of the 4nd argument (list) should be the same as either dimension of the first argument (matrix)');
  ASSERT(vectdim(WD) = 2*coldim(A),
    'the length of the 5nd argument (list) should be twice either dimension of the first argument (matrix)');
  p := [z1 - 1, z2 - 1, z1*z2 - 1];
  BM := evalm(B*diag(seq(1, i = 1 .. rowdim(A))));
  BD := map(factor, diag(seq(op([2*BM[i, i]*p[W[i][1]] / subs({z1 = z1^2, z2 = z2^2}, p[W[i][1]]]),
    2*BM[i, i]*p[W[i][2]] / subs({z1 = z1^2, z2 = z2^2}, p[W[i][2]]])), i = 1 .. rowdim(A)));
  ASSERT(IsLaurentMatrix(BD), 'the comparison scheme does not have necessary factors');
  Q := DiffScheme(A, W);
  Q2 := DiffScheme(evalm(2*Q), WD);
  DA := map(expand, map(factor, evalm(A - BM)));
  DQ := map(expand, map(factor, evalm(2*Q - BD)));
  res[gamma] := InfNnorm(Q, n);
  res[c] := InfNnorm(IdealDecomposeMatrix(DA, W, true), 1);
  res[gammaD] := InfNnorm(Q2, nD);
  res[cD] := InfNnorm(IdealDecomposeMatrix(DQ, WD, true), 1);
  eval(res)
end

```

Estimates for schemes with a comparison factor

Another type of simplification can be made for schemes with Laurent polynomials of the form $B(z_1, z_2)Q(z_1, z_2)$ (see complete conditions below).

LinearDecompose(A, i, j) Given a Laurent polynomial $A(z_1, z_2)$ known to be in the ideal generated by $z_1 - 1$ and $z_2 - 1$ find polynomials TA_1 and TA_2 such that $A = (z_1 - 1)TA_1 + (z_2 - 1)TA_2$, or $A = (z_1 z_2 - 1)TA_1 + (z_2 - 1)TA_2$ or $A = (z_1 - 1)TA_1 + (z_1 z_2 - 1)TA_2$; the expressions are derived from the expressions of Lemma 2.3 [CDM]. The decomposition is not unique; the expressions are asymmetric even in the first case; the choice of the pair of polynomials is given by the arguments i and j ; i corresponds to $z_1 - 1$, 2 to $z_2 - 1$ and 3 to $z_1 z_2 - 1$. The order matters; the last argument is used to request symmetrized decomposition with respect to $z_1 - 1$, and $z_2 - 1$; if one of i, j is 3 , the last argument is ignored. The function returns a list of 2 polynomials.

```

> LinearDecompose := proc(A::LaurentPolynom, i::integer, j::integer, symmetrize::boolean)
  local TA1, TA2, Q, p;
  global z1, z2;
  ASSERT(1 ≤ i and i ≤ 3 and 1 ≤ j and j ≤ 3 and i ≠ j,
    'second and third argument cannot coincide and should be in the range 1..3');
  ASSERT(simplify(subs({z1 = 1, z2 = 1}, A)) = 0, 'Laurent polynomial does not have root [1,1]');
  p[1] := z1 - 1;
  p[2] := z2 - 1;
  p[3] := z1*z2 - 1;
  Q[1, 2] := subs(z1 = 1, A);
  Q[1, 3] := subs({z2 = z1*z2, z1 = 1}, A);
  Q[2, 1] := subs(z2 = 1, A);
  Q[2, 3] := subs({z1 = z1*z2, z2 = 1}, A);
  Q[3, 1] := subs(z2 = 1 / z1, A);
  Q[3, 2] := subs(z1 = 1 / z2, A);
  if symmetrize and i = 1 and j = 2 then
    TA1 := factor(1*(A + Q[2, 1] - Q[1, 2]) / 2*(z1 - 1)); TA2 := factor(1*(A + Q[1, 2] - Q[2, 1]) / 2*(z2 - 1))
  elif symmetrize and i = 2 and j = 1 then

```

```

    TA2 := factor(1*(A + Q[2, 1] - Q[1, 2]) / 2*(z1 - 1)); TAI := factor(1*(A + Q[1, 2] - Q[2, 1]) / 2*(z2 - 1))
  else TAI := factor(Q[j, i] / p[i]); TA2 := factor((A - Q[j, i]) / p[j])
  fi;
  ASSERT(IsLaurentPolynom(TAI) and IsLaurentPolynom(TA2), [TAI, TA2]);
  if simplify(p[i]*TAI + p[j]*TA2 - A) ≠ 0 then print( 'decomposition error' ) fi;
  [TAI, TA2]
end

```

▣ **FactorizableConvergenceEstimates(A, n, nD, W, WD, B)**. Similar to **ComputeEstimates** but with additional assumptions on A and B. A should be a Laurent polynomial (not matrix) divisible by B; W and WD should have lengths 1 and 2 respectively, and if W_i are the Laurent polynomials corresponding to the components of W, and WD_i are the Laurent polynomials

of the differences corresponding to the components of WD, then B should be divisible by $\frac{W_i(z_1^2, z_2^2) WD_{i,j}(z_1, z_2)}{W_i(z_1, z_2) WD_{i,j}(z_1, z_2)}$; for

example, if $W_i = z_1 - 1$ and $W_{i,j} = z_1 z_2 - 1$, then B should be divisible by $(z_1 + 1)(z_1 z_2 + 1)$. The estimates for γ and γ_D calculated by this function are the same as the estimates calculated by **ComputeEstimates**; however, the estimates for c and cD tend to be better.

```

> FactorizableConvergenceEstimates := proc(
  A::LaurentPolynom, n::posint, nD::posint, W::IntegerVector2List, WD::IntegerVector2List, B::LaurentPolynom)
  local q, pplus, psq, a1, a2, a11, a12, a21, a22, b1, b2, b11, b12, b21, b22, tq, tq1, tq2, res;
  global z1, z2, gammaD, c, cD;
  ASSERT(vectdim(W) = 1, 'the length of the 4nd argument (list) should be 1');
  ASSERT(vectdim(WD) = 2, 'the length of the 5nd argument (list) should be 2');
  q := factor(A / B);
  ASSERT(IsLaurentPolynom(q), 'A should be divisible by B');
  pplus := [z1 + 1, z2 + 1, z1*z2 + 1];
  psq := [z1^2 - 1, z2^2 - 1, z1^2*z2^2 - 1];
  b1 := factor(B / pplus[W[1][1]]);
  b2 := factor(B / pplus[W[1][2]]);
  b11 := factor(b1 / pplus[WD[1][1]]);
  b12 := factor(b1 / pplus[WD[1][2]]);
  b21 := factor(b2 / pplus[WD[2][1]]);
  b22 := factor(b2 / pplus[WD[2][2]]);
  ASSERT(
    IsLaurentPolynom(b11) and IsLaurentPolynom(b12) and IsLaurentPolynom(b21) and IsLaurentPolynom(b22),
    'the comparison scheme Laurent polynomial is not divisible by some of the difference polynomials');
  tq := LinearDecompose(q - 1, W[1][1], W[1][2], true);
  tq1 := LinearDecompose(q - 1, WD[1][1], WD[1][2], true);
  tq2 := LinearDecompose(q - 1, WD[2][1], WD[2][2], true);
  a1 := factor(A / pplus[W[1][1]]);
  a2 := factor(A / pplus[W[1][2]]);
  res[gamma] := max(InfNnorm(matrix([[a1]]), n), InfNnorm(matrix([[a2]]), n));
  ASSERT(simplify(A - B - b1*tq[1]*psq[W[1][1]] - b2*tq[2]*psq[W[1][2]]) = 0, 'bad decomp');
  res[c] := InfNnorm(matrix([[b1*tq[1]]], 1) + InfNnorm(matrix([[b2*tq[2]]], 1));
  a11 := factor(2*a1 / pplus[WD[1][1]]);
  a12 := factor(2*a1 / pplus[WD[1][2]]);
  a21 := factor(2*a2 / pplus[WD[2][1]]);
  a22 := factor(2*a2 / pplus[WD[2][2]]);
  res[gammaD] := max(InfNnorm(matrix([[a11]]), nD), InfNnorm(matrix([[a12]]), nD),
    InfNnorm(matrix([[a21]]), nD), InfNnorm(matrix([[a22]]), nD));
  res[cD] := max(InfNnorm(matrix([[2*b11*tq1[1]]], 1) + InfNnorm(matrix([[2*b12*tq1[2]]], 1),
    InfNnorm(matrix([[2*b21*tq2[1]]], 1) + InfNnorm(matrix([[2*b22*tq2[2]]], 1));
  eval(res)
end

```

Results for specific schemes

Bilinear and Trilinear

This is just a sanity check; note that because we use difference schemes of a linear scheme as comparison scheme for derivatives, which is not continuous, we cannot infer C1-continuity from the fact that $cD = 0$. In general, we assume that C1-continuity is known, and we

are interested in the rate of approximation. To establish C1-continuity, one has to look only at γ_D

```
> Linear := (1+z)^2/2: Bilinear := evalm(diag(1)*(1/4)*(1 + z1^(-1))^2*(1 +
z2^(-1))^2*z1*z2):
> op(ConvergenceEstimates(Bilinear, 1,1,[[1,2]], [[1,2],[2,1]], Bilinear[1,1]));
```

$$\left[c = 0, cD = 0, \gamma = \frac{1}{2}, \text{gamma}D = 1 \right]$$

```
> op(FactorizableConvergenceEstimates(Bilinear[1,1], 1,1,[[1,2]], [[1,2],[2,1]],
Bilinear[1,1]));
```

$$\left[c = 0, cD = 0, \gamma = \frac{1}{2}, \text{gamma}D = 1 \right]$$

```
> TrilinearEstimate := ConvergenceEstimates(evalm(Trilinear * diag(1)),
1,1,[[1,2]], [[2,3],[3,1]], Trilinear) : op(op(TrilinearEstimate));
```

$$\left[c = 0, cD = 0, \gamma = \frac{1}{2}, \text{gamma}D = 1 \right]$$

3-directional box spline (Loop)

```
> Trilinear := (1/2)*(1+z1)*(1+z2)*(1+z1*z2)*z1^(-1)*z2^(-1):
ThreeDirCoeffs := array( -2..2, -2..2, [
[ 1/16, 1/8, 1/16, 0, 0 ],
[ 1/8, 3/8, 3/8, 1/8, 0 ],
[ 1/16, 3/8, 5/8, 3/8, 1/16 ],
[ 0, 1/8, 3/8, 3/8, 1/8 ],
[ 0, 0, 1/16, 1/8, 1/16 ]
]);
```

```
> ThreeDir := evalm( diag(1)* factor(MakePolynom(ThreeDirCoeffs)));
```

$$\text{ThreeDir} := \frac{1}{16} \frac{(1+z1)^2 (1+z2)^2 (1+z1 z2)^2}{z1^2 z2^2}$$

Because the Laurent polynomial has all three difference factors, and each is squared, we can use a variety of contraction functions.

As it is typically the case, the special function for factorizable polynomials gives better estimates for cD .

```
> op(ConvergenceEstimates(ThreeDir, 1,1,[[1,2]], [[3,2],[1,3]], Trilinear));
```

$$\left[c = \frac{1}{2}, cD = 1, \gamma = \frac{1}{2}, \text{gamma}D = \frac{1}{2} \right]$$

```
> op(ConvergenceEstimates(ThreeDir, 1,1,[[1,2]], [[1,2],[2,1]], Bilinear[1,1]));
```

$$\left[c = \frac{1}{2}, cD = \frac{5}{8}, \gamma = \frac{1}{2}, \text{gamma}D = \frac{1}{2} \right]$$

```
> op(ConvergenceEstimates(ThreeDir, 1,1,[[1,2]], [[1,3],[1,3]], Bilinear[1,1]));
```

$$\left[c = \frac{1}{2}, cD = \frac{3}{4}, \gamma = \frac{1}{2}, \text{gamma}D = \frac{1}{2} \right]$$

```
> op(FactorizableConvergenceEstimates(ThreeDir[1,1], 1,1,[[1,2]], [[3,2],[1,3]],
Trilinear));
```

$$\left[c = \frac{1}{2}, cD = \frac{1}{2}, \gamma = \frac{1}{2}, \text{gamma}D = \frac{1}{2} \right]$$

```
> ThreeDirEstimate := FactorizableConvergenceEstimates(ThreeDir[1,1], 1,1,[[1,2]],
```



```
[[1,2],[1,2]] ,Bilinear[1,1]): op(op(ThreeDirEstimate));
```

$$\left[c = \frac{1}{2}, cD = \frac{1}{2}, \gamma = \frac{1}{2}, \text{gammaD} = \frac{1}{2} \right]$$

Butterfly

```
> ButterflyCoeffs := proc(w) array( -3..3,-3..3, [
  [ 0, -w, -w, 0, 0, 0, 0 ],
  [ -w, 0, 2*w, 0, -w, 0, 0 ],
  [ -w, 2*w, 1/2, 1/2, 2*w, -w, 0 ],
  [ 0, 0, 1/2, 1, 1/2, 0, 0 ],
  [ 0, -w, 2*w, 1/2, 1/2, 2*w, -w ],
  [ 0, 0, -w, 0, 2*w, 0, -w ],
  [ 0, 0, 0, 0, -w, -w, 0 ]
]);
end:
> Butterfly := evalm( diag(1)* MakePolynom(ButterflyCoeffs(w)));
> ButterflyEstimate1 := ConvergenceEstimates(Butterfly, 1,1,[[1,2]], [[2,3],[3,1]]
,Trilinear): op(eval(ButterflyEstimate1));
  \left[ c = 8|w|, cD = 16|w|, \gamma = \max\left(9|w| + \left|\frac{1}{2} - 3w\right|, 4|w| + \frac{1}{2}\right), \text{gammaD} = \max(8|w| + |-8w + 1|, 16|w|) \right]
> map(simplify, op(subs(w = 1/16, eval(ButterflyEstimate1))));
  \left[ c = \frac{1}{2}, cD = 1, \gamma = \frac{7}{8}, \text{gammaD} = 1 \right]
> ButterflyEstimate2 := ConvergenceEstimates(Butterfly, 2,2,[[1,2]], [[2,3],[3,1]]
,Trilinear):
> assume(W > 0); additionally(W < 1/8);
  solve( simplify( subs( w = W, eval(ButterflyEstimate2[gammaD]))) < 1);
  \text{RealRange}\left(\text{Open}(0), \text{Open}\left(\frac{1}{12}\right)\right)
> op(map( eval, subs(w = 1/16, eval(ButterflyEstimate2))));
  \left[ c = \frac{1}{2}, cD = 1, \gamma = \frac{31}{64}, \text{gammaD} = \frac{7}{8} \right]
> ButterflyEstimate3 := ConvergenceEstimates(Butterfly, 3,3,[[1,2]], [[2,3],[3,1]]
,Trilinear):
> assume(W > 1/12); additionally(W < 1/8);
  simplify( subs( w = W, eval(ButterflyEstimate3[gammaD])));
max(456 W^3 + 20 W^2 + |-184 W^3 + 4 W - 20 W^2|,
-336 W^3 + 60 W^2 + |2 W + 432 W^3 - 48 W^2| + |-328 W^3 + 36 W^2| + |136 W^3 - 16 W^2|, 416 W^3 + 64 W^2,
408 W^3 + 12 W^2 - 2 W + |6 W - 68 W^2| + |-408 W^3 + 40 W^2|, 672 W^3 - 16 W^2 + 4 W, 24 W^2 - 216 W^3
+ |-2 W + 44 W^2 - 360 W^3| + |-16 W^2 + 160 W^3| + |488 W^3 - 96 W^2 + 6 W| + 2 W + |2 W + 216 W^3 - 52 W^2|,
224 W^3 + 16 W^2 + |880 W^3 + 12 W - 208 W^2| + 4 W + |1 - 24 W - 848 W^3 + 224 W^2|,
288 W^3 + 2 |120 W^3 - 2 W| + |-432 W^3 + 128 W^2 - 20 W + 1| + |-112 W^2 + 8 W + 288 W^3|,
-512 W^3 + 80 W^2 + 2 |-136 W^3 - 2 W + 28 W^2| + |32 W^2 - 16 W + 1 - 96 W^3| + |4 W - 56 W^2 + 336 W^3|,
264 W^3 + 4 W^2 + |-56 W^2 + 456 W^3 + 4 W| + |60 W^2 - 416 W^3 - 2 W|,
-288 W^3 + 2 |20 W^2 - 120 W^3 - 2 W| + 48 W^2 + |1 + 8 W^2 - 16 W + 432 W^3| + |-16 W^2 + 4 W - 288 W^3|,
264 W^3 + 16 W^2 + |-2 W + 88 W^2 - 584 W^3| + |640 W^3 + 6 W - 104 W^2|);
> op(map( eval, subs(w = 1/16, eval(ButterflyEstimate3))));
  \left[ c = \frac{1}{2}, cD = 1, \gamma = \frac{261}{1024}, \text{gammaD} = \frac{11}{16} \right]
```

Tensor product of quadratic splines (Doo-Sabin)

```
> Quadratic := expand((1/4)*(z^(-1) + 1)^3*z); QuadraticTensorSpline :=
evalm(diag(1)*(1/16)*(z1^(-1) + 1)^3*(z2^(-1) + 1)^3*z1*z2 );
```

$$Quadratic := \frac{1}{4} \frac{1}{z^2} + \frac{3}{4} \frac{1}{z} + \frac{3}{4} + \frac{1}{4} z$$

$$QuadraticTensorSpline := \left[\frac{1}{16} \left(1 + \frac{1}{zI} \right)^3 \left(1 + \frac{1}{z2} \right)^3 zI z2 \right]$$

```
> op(ConvergenceEstimates(QuadraticTensorSpline, 1,1,[[1,2]], [[1,2],[2,1]],
,Bilinear[1,1]));
```

$$\left[c = \frac{1}{2}, cD = \frac{3}{4}, \gamma = \frac{1}{2}, gammaD = \frac{1}{2} \right]$$

```
> op(FactorizableConvergenceEstimates(QuadraticTensorSpline[1,1], 1,1,[[1,2]],
[[1,2],[2,1]], Bilinear[1,1]));
```

$$\left[c = \frac{1}{2}, cD = \frac{3}{4}, \gamma = \frac{1}{2}, gammaD = \frac{1}{2} \right]$$

Tensor product of cubic splines (Camull-Clark)

```
> Cubic := (1/8)*(1+z^(-1))^4*z^2; TensorCubicSpline := evalm(diag(1)*(1/64)*(z1^(-1)
+ 1)^4*(z2^(-1) + 1)^4*z1^2*z2^2 );
```

$$Cubic := \frac{1}{8} \left(1 + \frac{1}{z} \right)^4 z^2$$

$$TensorCubicSpline := \left[\frac{1}{64} \left(1 + \frac{1}{zI} \right)^4 \left(1 + \frac{1}{z2} \right)^4 zI^2 z2^2 \right]$$

```
> op(ConvergenceEstimates(TensorCubicSpline, 1,1,[[1,2]], [[2,1],[1,2]],
,Bilinear[1,1]));
```

$$\left[c = \frac{1}{2}, cD = \frac{11}{16}, \gamma = \frac{1}{2}, gammaD = \frac{1}{2} \right]$$

```
> TensorCubicEstimate := FactorizableConvergenceEstimates(TensorCubicSpline[1,1],
1,1,[[1,2]], [[2,1],[1,2]], Bilinear[1,1]): op(op(TensorCubicEstimate));
```

$$\left[c = \frac{1}{2}, cD = \frac{1}{2}, \gamma = \frac{1}{2}, gammaD = \frac{1}{2} \right]$$

Tensor product of 4-point schemes (Kobbelt)

```
> FourPt := factor((-w*z^(-3) + (1/2+w)*z^(-1) + 1 + (1/2+w)*z - w*z^3));
TensorFourPt := matrix([[factor((-w*z1^(-3) + (1/2+w)*z1^(-1) + 1 + (1/2+w)*z1 -
w*z1^3)*( -w*z2^(-3) + (1/2+w)*z2^(-1) + 1 + (1/2+w)*z2 -w*z2^3 ))]]);
```

$$FourPt := -\frac{1}{2} \frac{(1+z)^2 (2z^4 w - 4wz^3 + 4z^2 w - z^2 - 4wz + 2w)}{z^3}$$

TensorFourPt :=

$$\left[\frac{1}{4} \right]$$

$$\frac{(1+zI)^2 (2zI^4 w - 4wzI^3 + 4zI^2 w - zI^2 - 4zI w + 2w) (1+z2)^2 (2wz2^4 - 4z2^3 w + 4z2^2 w - z2^2 - 4z2 w + 2w)}{z2^3 zI^3}$$

]

```
> TensorFourPtEstimate1 := ConvergenceEstimates(TensorFourPt, 1,1,[[1,2]],
[[1,2],[2,1]], Bilinear[1,1]);
```

```
TensorFourPtEstimate1 := table[
```

$$c = 4|w|^2 + 2\left|\frac{1}{2}w^2 + \frac{1}{4}w\right| + 2|w| + 2\left|\frac{1}{2}w^2 + \frac{3}{4}w\right| + 2\left|-\frac{1}{4}w - \frac{1}{2}w^2\right| + 2\left|-\frac{3}{4}w - \frac{1}{2}w^2\right|$$

$$cD = \max\left(5|w|^2 + 2\left|\frac{1}{2}w + w^2\right| + 2|w| + \left|\frac{3}{2}w + w^2\right| + 2\left|-w^2 + \frac{1}{2}w\right| + |w^2 + w| + \left|-w^2 - \frac{1}{2}w\right|, 4|w|, 8|w|^2 + 2\left|-w^2 - \frac{1}{2}w\right| + 3|w| + 2\left|-w^2 - \frac{3}{2}w\right|\right)$$

$$\gamma = \max\left(2|w| + \frac{1}{2}, 4|w|^2 + 2\left|-w^2 - \frac{1}{2}w\right| + |w| + 2\left|\frac{1}{2}w + \frac{1}{4}\right| + 2\left|\frac{1}{2}w + w^2\right|\right)$$

$$gammaD = \max\left(8|w|^2 + 4|w| + \frac{1}{2}, 8|w|^2 + 4|-2w^2 - w| + 2|4w^2 - w| + 2\left|-4w^2 - w + \frac{1}{2}\right|, 8|w|, 16|w|^2 + 4|4w^2 + 2w|, 4|w| + |1 - 4w|\right)$$

```

])
> op( map( eval, subs( w = 1/16, eval(TensorFourPtEstimate1)))));
      [ c = 13/32, cD = 31/64, gamma = 25/32, gammaD = 5/4 ]
> TensorFourPtEstimate2 := ConvergenceEstimates(TensorFourPt, 2,2,[[1,2]],
[[1,2],[2,1]],Bilinear[1,1]):
> op(map( eval, subs( w = 1/16, eval(TensorFourPtEstimate2)))));
      [ c = 13/32, cD = 31/64, gamma = 105/256, gammaD = 15/16 ]
> TensorFourPtEstimate3 := ConvergenceEstimates(TensorFourPt, 3,3,[[1,2]],
[[1,2],[2,1]],Bilinear[1,1]):
op(map( eval, subs( w = 1/16, eval(TensorFourPtEstimate3)))));
      [ c = 13/32, cD = 31/64, gamma = 425/2048, gammaD = 5/8 ]
> TensorFourPtEstimate1f := op(FactorizableConvergenceEstimates(TensorFourPt[1,1],
1,1,[[1,2]], [[2,1],[1,2]],Bilinear[1,1]));
TensorFourPtEstimate1f := [ c = 2 max( 2|w|, 2|w|^2 + 2|1/2 w^2 + 1/2 w| + 2|-1/2 w^2 - 1/2 w| )
cD = 4|w|^2 + 2|w| + max(4|w|^2 + 4|-w - w^2|, 4|w|),
gamma = max( 2|w| + 1/2, 4|w|^2 + 2|-w^2 - 1/2 w| + |w| + 2|1/2 w + 1/4| + 2|1/2 w + w^2| ), gammaD = max( 8|w|^2 + 4|w| + 1/2,
8|w|^2 + 4|-2 w^2 - w| + 2|4 w^2 - w| + 2|-4 w^2 - w + 1/2|, 8|w|, 16|w|^2 + 4|4 w^2 + 2 w|, 4|w| + |1 - 4 w| ) ]
> op(map( eval, subs( w = 1/16, eval(TensorFourPtEstimate1f)))));
      c = 9/32, cD = 27/64, gamma = 25/32, gammaD = 5/4

```

Code generation

Eigenstructure of the Butterfly Scheme

Denis Zorin, February 1998

This worksheet contains the symbolic part of the analysis of tangen plane continuity and C1-continuity of the Butterfly scheme. We compute the eigenvalues of the subdivision matrix with the maximal magnitude and the corresponding eigenvectors. For the valence $K = 3$, the largest eigenvalue is $1/4$, with 3 Jordan blocks: two of size 2 and one of size 3. For $K = 4, 5, 7$ the largest eigenvalues are in the first and last blocks of the DFT-transformed matrix, and have trivial Jordan blocks. For $K \geq 8$, the largest eigenvalues are in other blocks. We generate the C-code for the computationally intensive part of the analysis (analysis of the characteristic maps). The generated code uses functions from our wrapper class for f.p. numbers, encapsulating interval arithmetics.

+ Utilities

- Subdivision matrix

```
> assume( c >= -1); additionally( c <= -1); additionally (c, real); assume( K, integer);
```

```
additionally( K >= 3);
```

```
> Butterfly := matrix( [
  [ (1/2) + 4*w*c - 2*w*(2*c^2-1), 0, -w*(conjugate(omega) + 1), 0, 0, 0],
  [1, 0, 0, 0, 0, 0],
  [(1/2)*(1+ omega) - w*(conjugate(omega) + omega^2 ), -w*(1 + omega), 2*w, 0, 0, 0],
  [1/2 - 2*w*c, 1/2, 2*w*(1+conjugate(omega)), 0, -w, -w*conjugate(omega)],
  [1/2 + 2*w*omega, 2*w - w*omega, 1/2-w*conjugate(omega), 0, -w, 0],
  [(1/2)*omega + 2*w, 2*w*omega-w, 1/2 - w*omega, 0, 0, -w]]);
```

$$Butterfly := \begin{bmatrix} \frac{1}{2} + 4wc - 2w(2c^2 - 1) & 0 & -w(\bar{\omega} + 1) & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2}(1 + \omega) - w(\bar{\omega} + \omega^2) & -w(1 + \omega) & 2w & 0 & 0 & 0 \\ \frac{1}{2} - 2wc & \frac{1}{2} & 2w(\bar{\omega} + 1) & 0 & -w & -w\bar{\omega} \\ \frac{1}{2} + 2w\omega & 2w - w\omega & \frac{1}{2} - w\bar{\omega} & 0 & -w & 0 \\ \frac{1}{2}\omega + 2w & 2w\omega - w & \frac{1}{2} - w\omega & 0 & 0 & -w \end{bmatrix}$$

```
> Butterconst := { w = 1/16}; Buttervar := { omega = exp(2*I*Pi*m/K), c = cos(2*m*Pi/K)
};
```

$$Butterconst := \left\{ w = \frac{1}{16} \right\}$$

$$Buttervar := \left\{ c = \cos\left(2\frac{m\pi}{K}\right), \omega = e^{\left(2\frac{I\pi m}{K}\right)} \right\}$$

```
> ButterflyExpanded := map( simplify, subs( s^2 = 1-c^2, map( simplify, map( evalc,
  subs( { omega = c + s*I , op(Butterconst)}, eval(Butterfly))) )))
;
```

```
ButterflyExpanded :=
```

$$\begin{bmatrix} \frac{5}{8} + \frac{1}{4}c - \frac{1}{4}c^2 & 0 & -\frac{1}{16}c - \frac{1}{16} + \frac{1}{16}Is & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{16} + \frac{7}{16}c - \frac{1}{8}c^2 + \frac{9}{16}Is - \frac{1}{8}Ics & -\frac{1}{16} - \frac{1}{16}c - \frac{1}{16}Is & \frac{1}{8} & 0 & 0 & 0 \\ \frac{1}{2} - \frac{1}{8}c & \frac{1}{2} & \frac{1}{8}c + \frac{1}{8} - \frac{1}{8}Is & 0 & \frac{-1}{16} & -\frac{1}{16}c + \frac{1}{16}Is \\ \frac{1}{2} + \frac{1}{8}c + \frac{1}{8}Is & \frac{1}{8} - \frac{1}{16}c - \frac{1}{16}Is & \frac{1}{2} - \frac{1}{16}c + \frac{1}{16}Is & 0 & \frac{-1}{16} & 0 \\ \frac{1}{2}c + \frac{1}{2}Is + \frac{1}{8} & \frac{1}{8}c + \frac{1}{8}Is - \frac{1}{16} & \frac{1}{2} - \frac{1}{16}c - \frac{1}{16}Is & 0 & 0 & \frac{-1}{16} \end{bmatrix}$$

Because the matrix has block-diagonal structure, we have to compute only the eigenvalues of the subblocks on the diagonal.

```
> Butterfly00 := submatrix( ButterflyExpanded, 1..3,1..3):
> Butterfly11 := submatrix( ButterflyExpanded, 4..6,4..6):
> Butterfly10 := submatrix( ButterflyExpanded, 4..6,1..3):
>
```

Introduce new variables, cs and ss, for $\cos\left(\frac{m\pi}{K}\right)$ and $\sin\left(\frac{m\pi}{K}\right)$.

```
> Cre := diag( 1, 1, cs - I*ss );
```

$$Cre := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & cs - I ss \end{bmatrix}$$

Reduce the first subblock to a real matrix using a coordinate transform. Eigenvalues do not change.

```
> Butter00re := map( simplify, subs( ss^2 = 1 - cs^2, map( expand, map( simplify, subs(
{ s^2 = 1 - c^2 }, subs( { c = 2*cs^2 - 1, s = 2*cs*ss }, map(simplify, map( evalc, map(
simplify, evalm( Cre &* eval(Butterfly00) &* inverse(Cre))))))))));
```

$$Butter00re := \begin{bmatrix} \frac{1}{8} + \frac{3}{2}cs^2 - cs^4 & 0 & -\frac{1}{8}cs \\ 1 & 0 & 0 \\ -\frac{1}{2}cs^3 + \frac{11}{8}cs & -\frac{1}{8}cs & \frac{1}{8} \end{bmatrix}$$

Analysis of the behavior of the eigenvalues

Our goal is to determine the expressions for the eigenvalues of the largest magnitude, excluding 1, and show that for valences > 8 these eigenvalues are not in the 1st and last blocks of the DFT-transformed subdivision matrix. With some additional easily provable assumptions, this means that the Butterfly scheme is not C1 for these valences. We also explicitly compute the eigenvalues for $K = 3$.

0-th block

The block corresponding to $m = 0$ is present in every matrix; if the eigenvalues of some other block are greater than $1/4$, this block is not dominant.

```
> jordan(subs( cs = 1, eval(Butter00re)));
```

$$\begin{bmatrix} \frac{1}{4} & 1 & 0 \\ 0 & \frac{1}{4} & 1 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}$$

Characteristic polynomial of the first subblock and its discriminant

The eigenvalues of the second subblock are 0, and $-1/16$; we will see that the first subblock always has larger eigenvalues.

Characteristic polynomial

```
> ButterCharpoly := subs( cs = sqrt(d), collect( subs( cos(m*Pi/K) = c, expand(
charpoly(Butter00re,lambda),trig)), lambda));
```

$$\text{ButterCharpoly} := \lambda^3 + \left(-\frac{1}{4} + d^2 - \frac{3}{2}d\right)\lambda^2 + \left(\frac{23}{64}d + \frac{1}{64} - \frac{3}{16}d^2\right)\lambda - \frac{1}{64}d$$

```
> re := coeff( ButterCharpoly, lambda^2); se := coeff( ButterCharpoly, lambda); te :=
rem(ButterCharpoly, lambda, lambda);
```

$$re := -\frac{1}{4} + d^2 - \frac{3}{2}d$$

$$se := \frac{23}{64}d + \frac{1}{64} - \frac{3}{16}d^2$$

$$te := -\frac{1}{64}d$$

Reduce the characteristic polynomial; use $d = c^2$ as the parameter.

```
> ButterCharpolyReduced := collect(simplify(subs( lambda = mu - re/3,
ButterCharpoly)), mu);
```

ButterCharpolyReduced :=

$$\mu^3 + \left(-\frac{1}{192} + d^3 - \frac{37}{48}d^2 + \frac{7}{64}d - \frac{1}{3}d^4\right)\mu + \frac{1}{768}d + \frac{55}{1152}d^2 + \frac{73}{144}d^4 - \frac{19}{64}d^3 + \frac{2}{27}d^6 - \frac{1}{3}d^5 + \frac{1}{6912}$$

```
> pe := coeff(ButterCharpolyReduced, mu); qe := rem(ButterCharpolyReduced, mu, mu);
```

$$pe := -\frac{1}{192} + d^3 - \frac{37}{48}d^2 + \frac{7}{64}d - \frac{1}{3}d^4$$

$$qe := \frac{1}{768}d + \frac{55}{1152}d^2 + \frac{73}{144}d^4 - \frac{19}{64}d^3 + \frac{2}{27}d^6 - \frac{1}{3}d^5 + \frac{1}{6912}$$

Find the discriminant and determine its sign.

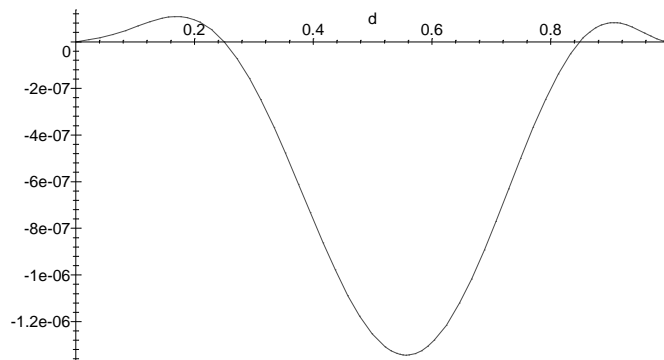
```
> Discr := simplify( (pe/3)^3 + (qe/2)^2);
```

$$\text{Discr} := \frac{1}{2359296}d - \frac{19}{3538944}d^2 - \frac{479}{442368}d^4 + \frac{1123}{7077888}d^3 - \frac{1369}{442368}d^6 + \frac{299}{110592}d^5 + \frac{91}{55296}d^7 - \frac{1}{3072}d^8$$

```
> Discr := factor(subs( w = 1/16, Discr));
```

$$\text{Discr} := -\frac{1}{7077888}d(4d-1)(576d^4 - 1616d^3 + 976d^2 - 20d + 3)(d-1)^2$$

```
> plot(Discr, d = 0..1);
```

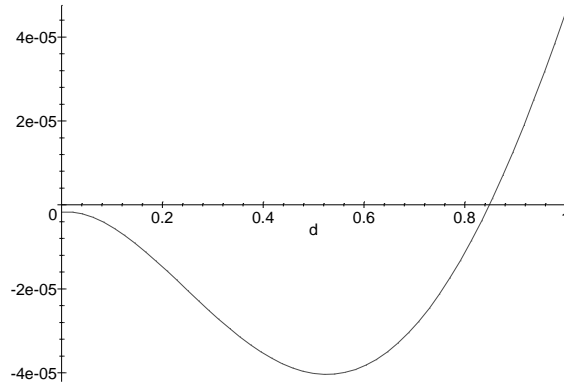


Pull out the degree 4 factor responsible for one of the roots on 0..1

```
DiscrFactor4 := factor(Discr/(lcoeff(Discr)*(d-1/4)*(d-1)^2*d));
```

$$\text{DiscrFactor4} := -\frac{1}{3072}d^4 + \frac{101}{110592}d^3 - \frac{61}{110592}d^2 + \frac{5}{442368}d - \frac{1}{589824}$$

```
> plot(DiscrFactor4, d=0..1);
```



Find the interesting root

```
> DiscrRoot := solve( { DiscrFactor4 = 0, d <= 1, d >= 0 }, d );
```

$$\text{DiscrRoot} := \left\{ d = \frac{101}{144} + \frac{1}{144} \sqrt{\%2} - \frac{1}{144} \sqrt{\frac{8690 \%1^{1/3} \sqrt{\%2} - 12 \sqrt{\%2} \%1^{2/3} - 657264 \sqrt{\%2} + 312154 \%1^{1/3}}{\%1^{1/3} \sqrt{\%2}}} \right\}$$

$$\%1 := 13061314 + 5238 \sqrt{229017}$$

$$\%2 := \frac{4345 \%1^{1/3} + 12 \%1^{2/3} + 657264}{\%1^{1/3}}$$

```
> DiscrRoot := op(2,op(DiscrRoot));
```

$$\text{DiscrRoot} := \frac{101}{144} + \frac{1}{144} \sqrt{\%2} - \frac{1}{144} \sqrt{\frac{8690 \%1^{1/3} \sqrt{\%2} - 12 \sqrt{\%2} \%1^{2/3} - 657264 \sqrt{\%2} + 312154 \%1^{1/3}}{\%1^{1/3} \sqrt{\%2}}}$$

$$\%1 := 13061314 + 5238 \sqrt{229017}$$

$$\%2 := \frac{4345 \%1^{1/3} + 12 \%1^{2/3} + 657264}{\%1^{1/3}}$$

```
> evalf(DiscrRoot);
```

.8486812039

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8486812039; these are the only values for which the matrix may have nontrivial Jordan blocks. The last case does not occur in the cases which are of interest to us. In the first 3 cases the Jordan normal form can be found explicitly.

☐ The case of three real roots

The discriminant is positive on $0..1/4$ and on $\text{DiscrRoot}..1$, negative on $1/4..\text{DiscrRoot}$

Compute the solutions when the discriminant is negative and, therefore, there are 3 real roots

```
> R := sqrt(-factor(pe)/3):
> phi := arccos( qe/(2*R^3)):
> r1 := -2*R*cos(phi/3)-re/3: r2 := -2*R*cos(phi/3 + 2*Pi/3)-re/3: r3 :=
-2*R*cos(phi/3 + 4*Pi/3)-re/3:
```

The product of the roots is $-d/64$, which is negative; therefore, either all three are positive, or two are negative. 0 is never a root, except when $d = 0$

The roots cannot be equal on the interval where the discriminant does not change sign; we can figure out the largest one on the whole interval by evaluating them at a single value of d . We conclude that all three roots are always positive and the largest one is the second.

$d = 1/4..\text{DiscrRoot}$. We use interval arithmetics to guarantee correctness.

```
> InvervalRoots := map( unapply('inapply'(x,d),x), [r1,r2,r3] );
> eval(InvervalRoots(1/2));
```

[[.08499852729, .08499853961], [.4690415185, .4690415259], [.1959599360, .1959599530]]

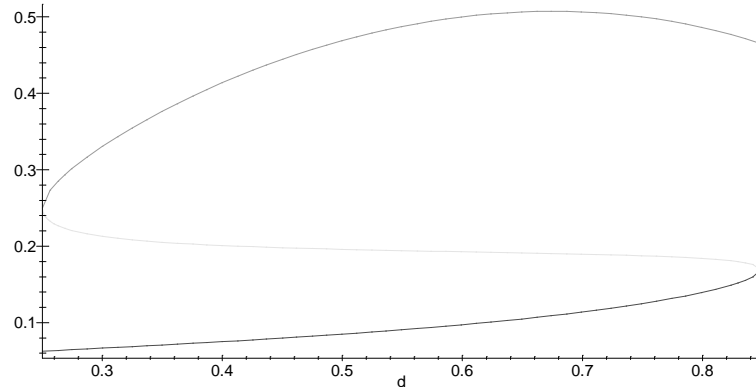
We conclude that the second root is the largest.

```
> AbsDominantEV1 := r2;
```

```

AbsDominantEVI := 1/12*sqrt((d-1)(64*d^3-128*d^2+20*d-1))
sin(1/3*arccos(6912*(1/768*d+55/1152*d^2+73/144*d^4-19/64*d^3+2/27*d^6-1/3*d^5+1/6912)
/((d-1)(64*d^3-128*d^2+20*d-1))^(3/2))+1/6*pi)+1/12-1/3*d^2+1/2*d
[[.08499852699,.08499853971],[.4690415184,.4690415260],[.1959599356,.1959599533]]
> plot([r1, r2, r3], d = 1/4..DiscrRoot);

```



■ The case of one real root

■ First case: The interval 0..1/4

We show that for d in this interval, all roots are $< 1/4$; we will see that there is always an eigenvalue $> 1/4$ elsewhere. Therefore, the roots on this interval are irrelevant. There is only one real root; if at a point x the value of the polynomial is positive, than the magnitude of the real root is less than x . We see that the char. polynomial is positive at $1/4$ for $d = 0..1/4$, and negative for $d = 1..1/4$. For any $K > 3$, $\frac{1}{4} < \cos\left(\frac{\pi}{K}\right)$, therefore, there is a real eigenvalue of magnitude greater than $1/4$.

```
> solve(subs(lambda = 1/4, ButterCharpoly) > 0);
```

$$\text{RealRange}\left(-\infty, \text{Open}\left(\frac{1}{4}\right)\right) \text{RealRange}(\text{Open}(1), \infty)$$

Now build an equation for the square of the magnitude of the other two roots; it is a cubic equation again:

```
> rsq := -se; ssq := expand(te*re); tsq := -te*te;
```

$$rsq := -\frac{23}{64}d - \frac{1}{64} + \frac{3}{16}d^2$$

$$ssq := \frac{1}{256}d - \frac{1}{64}d^3 + \frac{3}{128}d^2$$

$$tsq := -\frac{1}{4096}d^2$$

Use the same approach: verifying that all solutions are $< 1/16$ on $d=1..1/4$

```
> solve(subs(x = 1/16, x^3 + rsq*x^2 + ssq*x + tsq) > 0);
```

$$\text{RealRange}\left(-\infty, \text{Open}\left(\frac{1}{4}\right)\right) \text{RealRange}\left(\text{Open}\left(\frac{3}{4}\right), \text{Open}(1)\right)$$

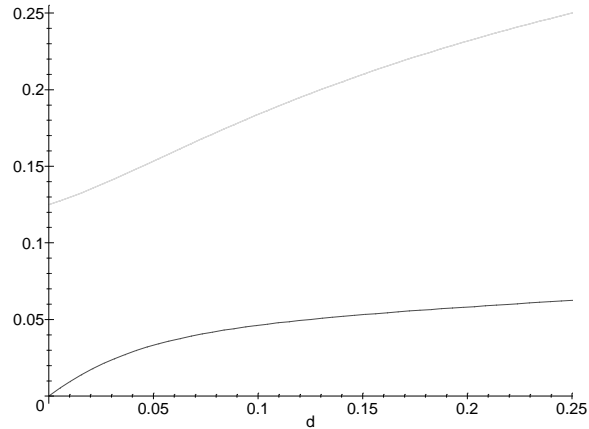
For plots, get the expressions for the roots

```
> R := signum(qe)*sqrt(-pe/3): phi := arccosh(abs(qe)/(2*abs(R)^3));
```

```
> r := -2*R*cosh(phi/3)-re/3: c1 := R*(cosh(phi/3) + I*sqrt(3)*sinh(phi/3))-re/3:
```

```
  c2 := R*(cosh(phi/3) + I*sqrt(3)*sinh(phi/3)) -re/3:
```

```
> plot([abs(r), abs(c1), abs(c2)], d=0..1/4);
```

Second case: interval DiscrRoot..1

In this case, we show that the real root is the largest; we have already observed that on $3/4 \dots 1$ the magnitude of the complex roots (when there are complex roots) is $< 1/4$; hence, it is sufficient to show that the real root is greater than $1/4$. But we have seen already, that the characteristic polynomial is negative at $1/4$ for $d > 1/4$. Therefore, the real root $> 1/4$.

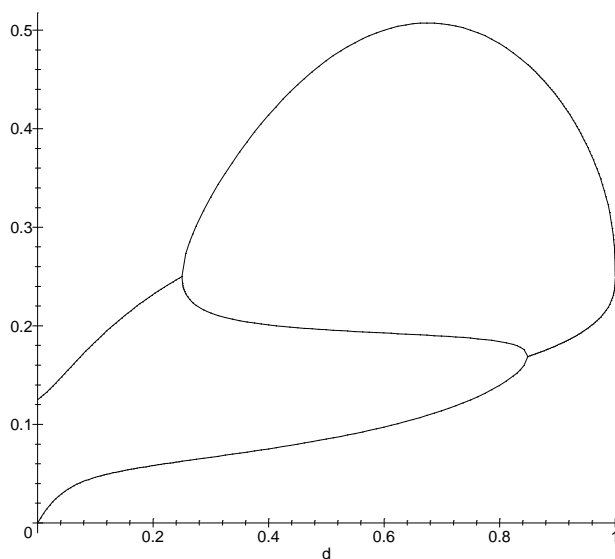
```
> R := sqrt(-pe/3): phi := arccosh( abs(qe)/(2*abs(R)^3): AbsDominantEV2 :=
  2*R*cosh(phi/3)-re/3;
```

$$AbsDominantEV2 := \frac{1}{12} \sqrt{1 - 192 d^3 + 148 d^2 - 21 d + 64 d^4}$$

$$\cosh\left(\frac{1}{3} \arccosh\left(6912 \frac{\left|\frac{1}{768}d + \frac{55}{1152}d^2 + \frac{73}{144}d^4 - \frac{19}{64}d^3 + \frac{2}{27}d^6 - \frac{1}{3}d^5 + \frac{1}{6912}\right|}{\left|1 - 192 d^3 + 148 d^2 - 21 d + 64 d^4\right|^{3/2}}\right)\right) + \frac{1}{12} - \frac{1}{3}d^2 + \frac{1}{2}d$$

Plot of all roots

```
> display(plot([abs(r),abs(c1)], d = 0..1/4, color=black),
  plot([r1,r2,r3],d=1/4..DiscrRoot, color=black),
  plot([abs(r),abs(c1)],d=DiscrRoot..1.000001, color=black));
```



▣ The magnitude of the subdominant eigenvalue decreases for d from approximately 0.67600423 to 1

To show that the subdominant eigenvalues of the Butterfly scheme are not in the correct block for sufficiently large valences, we show that the largest eigenvalue decreases as a function of d near 1. To do this, we find the sign of the derivative; it is sufficient to evaluate the derivative at a single point, and to show that the derivative is not zero anywhere on an interval. Let

$y(d)$ be the root as a function of d . Then differentiating the equation $y^3 + r(d)y^2 + s(d)y + t(d) = 0$, and setting $\frac{\partial}{\partial d}y$ to zero, we observe that at a point d_0 where the derivative is zero, the value $y(d_0)$ satisfies the equation

$\left(\frac{\partial}{\partial d}r\right)y^2 + \left(\frac{\partial}{\partial d}s\right)y + \left(\frac{\partial}{\partial d}t\right) = 0$. It is sufficient to show that for d on a given interval that the largest root of the original equation is not the root of the equation with differentiated coefficients.

```
> diffCharpoly := diff(re,d)*y^2 + diff(se,d)*y + diff(te,d);
```

$$\text{diffCharpoly} := \left(2d - \frac{3}{2}\right)y^2 + \left(\frac{23}{64} - \frac{3}{8}d\right)y - \frac{1}{64}$$

This is just a quadratic equation and we can immediately determine when it has no roots:

```
> solve( coeff(diffCharpoly, y)^2 - 4*rem(diffCharpoly,y,y)*coeff(diffCharpoly,y^2) < 0,d);
```

$$\text{RealRange}\left(\text{Open}\left(\frac{29}{72}\right), \text{Open}\left(\frac{5}{8}\right)\right)$$

We consider the interval $5/8..1$;

On this interval there is in fact a point where the magnitude of the largest root of the characteristic polynomial is maximal. It is useful to find it more precisely. We use Groebner bases package to eliminate y from the system of two equations and find a polynomial equation for d ; **finduni** finds the minimal univariate polynomial in the ideal generated by the two polynomials:

```
> dEquation := finduni( d, [subs( lambda = y, ButterCharpoly), diffCharpoly]);
```

$$dEquation := 9 - 48d + 332d^2 - 960d^3 + 1152d^4 - 512d^5$$

Now find a rational interval for d ; **realroot** provides us with guaranteed bounds on all real roots. Luckily, there is a single real root:

```
> IntervalDominantMax := op(realroot( dEquation, 1/10^7));
```

$$\text{IntervalDominantMax} := \left[\frac{11341469}{16777216}, \frac{5670735}{8388608}\right]$$

```
> map(evalf, IntervalDominantMax);
```

$$[.67600423097610473633, .67600429058074951172]$$

Evaluating the derivative at a point, we get a negative value; we conclude that the derivative is negative for d greater than the value above.

We use the algebraic value of the root returned by solve, rather than transcendental equation, because interval inapply does not work properly for hyperbolic functions.

```
> IntervDeriv := inapply( diff(op(1, [solve( ButterCharpoly, lambda)]),d),d):
eval(IntervDeriv(0.9));
```

$$[-.77122129552760084746, -.77122129552759906668]$$

We have shown that the magnitude of the largest eigenvalue decreases from approximately 0.6760043 to 1.

▣ Valence 3

In this case eigenvalue $1/4$ is the largest and has three identical Jordan blocks.

Block 0

```
> jordan(subs( cs = 1, eval(Butter00re)));
```

$$\begin{bmatrix} \frac{1}{4} & 1 & 0 \\ 0 & \frac{1}{4} & 1 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}$$

Block 1

```
> jordan( subs( cs = 1/2, eval(Butter00re)));
```

$$\begin{bmatrix} \frac{1}{16} & 0 & 0 \\ 0 & \frac{1}{4} & 1 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}$$

Block 2

```
> jordan( subs( cs = -1/2, eval(Butter00re)));
```

$$\begin{bmatrix} \frac{1}{16} & 0 & 0 \\ 0 & \frac{1}{4} & 1 \\ 0 & 0 & \frac{1}{4} \end{bmatrix}$$

Valences 4,5

K = 4; Check which formulas to use.

```
> inapply(DiscrRoot); inapply(cos(Pi/7)^4); inapply(cos(2*Pi/4)^2);
```

```
( ) → [.84868120391246120009, .84868120391246120352 ]
```

```
( ) → [.65892978418482746461, .65892978418482746486 ]
```

0

The largest eigenvalue is in the first block.

```
> inapply(subs( d = cos(Pi/4)^2, AbsDominantEV1));
```

```
( ) → [.46904152209925298145, .46904152209925298172 ]
```

```
> inapply(subs( d = cos(Pi/2)^2, AbsDominantEV1 ));
```

```
( ) → [.12499999999999999998, .12500000000000000002 ]
```

K = 5;

```
> inapply(DiscrRoot); inapply(cos(Pi/5)^2); inapply(cos(2*Pi/5)^2);
```

```
( ) → [.84868120391246120009, .84868120391246120352 ]
```

```
( ) → [.65450849718747371183, .65450849718747371227 ]
```

```
( ) → [.095491502812526287866, .095491502812526288029 ]
```

```
> inapply(subs( d = cos(Pi/5)^2, AbsDominantEV1));
```

```
( ) → [.50667561139380648476, .50667561139380649194 ]
```

We do not have to check the eigenvalue of the second block: for it, $d < 1/4$, therefore, the magnitude of the largest eigenvalue is also less than $1/4$.

For valence greater than 7, the eigenvalue of the block with $m = 1$ is not the largest

We have established that the magnitude of the largest eigenvalue decreases as the function of d when $d > 0.6700423 = d_0$; if

$$d = \cos\left(\frac{m\pi}{K}\right)^2 > d_0$$

for $m = 2$, for $K > 6$ we can conclude that the eigenvalue for $m = 2$ is greater than the eigenvalue for $m = 1$. This is the case for $K > 10$:

```
> x := inapply(2*Pi/K, K): eval(Interval_Integerpower(Interval_cos(x(1)), 2));
```

```
[.70770750650094321267, .70770750650094321286 ]
```

For values between 7 and 10, have to check one by one.

K = 7. Check which formulas to use. Note that for $m = 2$ the value is below d_0 , so we do not have to check the other values of m .

```
> inapply(DiscrRoot); inapply(cos(Pi/7)^2); inapply(cos(2*Pi/7)^2);
```

```
( ) → [.84868120391246120009, .84868120391246120352 ]
```

```
( ) → [.81174490092936676519, .81174490092936676535 ]
```

```
( ) → [.38873953302184279774, .38873953302184279798 ]
```

In this case, the largest eigenvalue is still in the first block:

```
> inapply(subs( d = cos(Pi/7)^2, AbsDominantEV1));
      ( ) → [.48191070141255089859, .48191070141255090538 ]
> inapply(subs( d = cos(2*Pi/7)^2, AbsDominantEV1));
      ( ) → [.40611653561495260677, .40611653561495260912 ]
```

K = 8

```
> inapply(DiscrRoot); inapply(cos(Pi/8)^2); inapply(cos(2*Pi/8)^2);
      ( ) → [.8486811866, .8486812214 ]
      ( ) → [.8535533896, .8535533917 ]
      ( ) → [.4999999998, .5000000002 ]
```

For $d_0 < d$, we cannot use the transcendental expression -- intpak does not have arccosh; rather than expressing it using logarithm, we use the algebraic expression, which works because the discriminant is positive:

```
> inapply(subs( d = cos(Pi/8)^2, op(1, [solve(ButterCharpoly, lambda)])));
      ( ) → [.46241776728303719789, .46241776728303860919 ]
```

We see that in this case the eigenvalue of the second block is larger:

```
> inapply(subs( d = cos(2*Pi/8)^2, AbsDominantEV1));
      ( ) → [.46904152209925298145, .46904152209925298172 ]
```

K = 9; same result as for 8:

```
> inapply(DiscrRoot); inapply(cos(Pi/9)^2); inapply(cos(2*Pi/9)^2);
      ( ) → [.84868120391246120009, .84868120391246120352 ]
      ( ) → [.88302222155948901753, .88302222155948901768 ]
      ( ) → [.58682408883346517429, .58682408883346517455 ]
> inapply(subs( d = cos(Pi/9)^2, op(1, [solve(ButterCharpoly, lambda)])));
      ( ) → [.44442935560347137503, .44442935560347191330 ]
> inapply(subs( d = cos(2*Pi/9)^2, AbsDominantEV1));
      ( ) → [.49729407293962378849, .49729407293962379215 ]
```

K = 10, same as for 8 and 9:

```
> inapply(DiscrRoot); inapply(cos(Pi/10)^2); inapply(cos(2*Pi/10)^2);
      ( ) → [.84868120391246120009, .84868120391246120352 ]
      ( ) → [.90450849718747371190, .90450849718747371220 ]
      ( ) → [.65450849718747371183, .65450849718747371227 ]
> inapply(subs( d = cos(Pi/10)^2, op(1, [solve(ButterCharpoly, lambda)])));
      ( ) → [.42859991276733051144, .42859991276733166709 ]
> inapply(subs( d = cos(2*Pi/10)^2, AbsDominantEV1));
      ( ) → [.50667561139380648476, .50667561139380649194 ]
```

Summary of the eigenvalue analysis

For $K = 3$, the largest eigenvalue is $1/4$ and has multiplicity 7, with 2 blocks of size 2 and one block of size 3. For $K = 4..7$ the largest eigenvalue has multiplicity 2 and corresponds to the 1st and the last block. For $K > 7$, the largest eigenvalues are not in the 1st and last blocks.

Eigenvectors

Find the eigenvectors in two steps. Using the special structure of the matrix $\begin{bmatrix} B_{0,0} & 0 \\ B_{1,0} & B_{1,1} \end{bmatrix}$, and the fact that we are interested in the eigenvectors which are also eigenvalues of the subblock $B_{0,0}$, we find the eigenvector as $[v_0, -(B_{1,1} - \lambda I)^{-1} B_{1,0} v_0]$, where v_0 is the eigenvector of $B_{0,0}$. We also use the fact that in all cases of interest, the eigenvalues of $B_{0,0}$ are not eigenvalues of $B_{1,1}$.

Compute an eigenvector of $B_{0,0}$; check first that the two second lines of the matrix are always independent; the second component

of the cross product is not zero, because $\frac{1}{4} \leq \lambda$:

```
> crossprod( row( evalm( subs( Butterfly00), evalm(Butterfly00)) - lambda* &*() ), 2),
row( evalm( Butterfly00 - lambda* &*() ), 3));
```

$$\left[-\lambda \left(\frac{1}{8} - \lambda \right), \lambda - \frac{1}{8}, -\frac{1}{16} - \frac{1}{16}c - \frac{1}{16}Is + \lambda \left(\frac{9}{16} + \frac{7}{16}c - \frac{1}{8}c^2 + \frac{9}{16}Is - \frac{1}{8}Ics \right) \right]$$

Compute the first part of the vector:

```
> v0 := subs( _t[1] = 1, map(simplify, linsolve(
submatrix ( evalm( Butterfly00- lambda*&*() ), 2..3, 1..3), [0,0] )));
```

>

$$v0 := \left[\lambda, 1, -\frac{1}{2} \frac{I(-I - Ic + s - 9s\lambda + 2sc\lambda + 9I\lambda - 2I\lambda c^2 + 7I\lambda c)}{-1 + 8\lambda} \right]$$

```
> Butter11lambda := evalm( Butterfly11 - lambda* &*());
```

>

$$Butter11lambda := \begin{bmatrix} -\lambda & \frac{-1}{16} & -\frac{1}{16}c + \frac{1}{16}Is \\ 0 & -\frac{1}{16} - \lambda & 0 \\ 0 & 0 & -\frac{1}{16} - \lambda \end{bmatrix}$$

```
> v1 := map( simplify, subs( { s^3 = s*(1-c^2), s^2 = 1- c^2}, map( expand, evalm( -
inverse( Butter11lambda ) &* Butterfly10 &* v0 ))));
```

$$v1 := \left[-\frac{1}{16} \frac{-3c - 1 - 1024\lambda^3 + 64\lambda^2 c^2 + 165\lambda - 208\lambda^2 c + 45\lambda c + 256\lambda^3 c - 1120\lambda^2 - 10\lambda c^2}{(-1 + 8\lambda)(1 + 16\lambda)\lambda}, \right. \\ \left. -\frac{1}{2} \frac{-79\lambda - 128\lambda^2 - 29\lambda c - 32\lambda^2 c - 61Is\lambda - 32Is\lambda^2 + 11 + 5c + 7Is + 18Is c\lambda + 14\lambda c^2}{(-1 + 8\lambda)(1 + 16\lambda)}, \frac{1}{2} \frac{(-11c + 2c^2 - 11Is + 61\lambda + 128\lambda^2 c + 61\lambda c + 2Ics - 32Is c\lambda + 79Is\lambda + 4c^3\lambda + 32\lambda^2 + 128Is\lambda^2 + 4Ic^2 s\lambda - 32\lambda c^2 - 7)}{((-1 + 8\lambda)(1 + 16\lambda))} \right]$$

Put the two parts of the vector together and simplify notation

```
> ButterEigenvect := array( map( simplify, [seq( v0[i], i=1..3), seq(v1[i], i=1..3)]));
```

Check if the expression makes sense for $K = 6$

```
> map(expand, subs( {lambda = 1/2, c = 1/2, s = sqrt(3)/2}, eval(ButterEigenvect)));
```

$$\left[\frac{1}{2}, 1, \frac{3}{4} + \frac{1}{4}I\sqrt{3}, \frac{3}{2}, \frac{5}{4} + \frac{1}{4}I\sqrt{3}, 1 + \frac{1}{2}I\sqrt{3} \right]$$

Code generation

Three functions are generated (same as for other schemes):

Float Eigenvalue(int K) computes the eigenvalues,

void EigenvectorReal(Float c, Float lambda, Float* EvRe) initializes an array for the real part of the complex eigenvector, void EigenvectorImaginary(Float c, Float* lambda, Float* EvIm) initializes the array for the complex part.

Memory for arrays should be allocated by the calling function.

The output is written to a file; if the name is 'default', it is written to the standard output (warning: for some reason, writing to standard output is terribly slow; writing to a file and then looking at it in an editor is much more efficient. All functions use Float as the name of the class for the interval numbers.

It is assumed to have explicit casts from 64-bit integers, standard arithmetics operations, and macros FR and Fdiv, (see ConvertToFloat for details).

```
> OutputFile := `butterfly.cpp`:
```

```
> MakeClassHeader( OutputFile, `Butterfly`, 2,4,3, RegButterfly):
```

Code generation for eigenvalues

To show that the scheme produces C1 surfaces for valences 4,5,7, and not C1 surfaces for other valences we need expressions for eigenvalues of the first block of the DFT-transformed subdivision matrix ($m = 1$).

We generate two functions, one to be used for valences 4,5,7; the other for larger valences.

+ `ComputeEigenvalues(N,eps,fname)` Numerically compute eigenvalues for a range, and write a function with a large table into a file.

Although we have computed explicit formulas above, they are numerically unstable for d close to 1 (for large K); the simplest solution is to

precompute the largest eigenvalue numerically with `verified`; we use the fact that the largest eigenvalues is in the interval $1/4 .. 1$ in the range of interest. This function computes eigenvalues up to valence N , verifying that the precision is no less than `eps`.

+ `Generate eigenvalues`

+ `Code generation for eigenvectors`

▣ Modified Butterfly scheme

The Modified Butterfly scheme by construction always has the subdominant eigenvalue $1/2$ in the first block of the DFT-transformed subdivision matrix.

Thus, we only need to compute the eigenvectors for the characteristic map analysis. We do not assume that $w = 1/16$ here.

▣ `Compute the complex eigenvector`

```
> ModButter := matrix( [
  [ 1/2, 0, 0,0,0,0],
  [1,0,0,0,0,0],
  [(1/2)*(1+ omega) - w*(conjugate(omega) + omega^2 ), -w*(1 + omega),2*w,0,0,0],
  [1/2 - 2*w*c,1/2,2*w*(1+conjugate(omega)),0, -w, -w*conjugate(omega)],
  [1/2 + 2*w*omega, 2*w - w*omega,1/2-w*conjugate(omega),0,-w,0],
  [(1/2)*omega + 2*w,2*w*omega-w, 1/2 - w*omega,0,0,-w]]);
```

$$ModButter := \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} + \frac{1}{2}\omega - w(\bar{\omega} + \omega^2) & -w(1 + \omega) & 2w & 0 & 0 & 0 \\ \frac{1}{2} - 2wc & \frac{1}{2} & 2w(\bar{\omega} + 1) & 0 & -w & -w\bar{\omega} \\ \frac{1}{2} + 2w\omega & 2w - w\omega & \frac{1}{2} - w\bar{\omega} & 0 & -w & 0 \\ \frac{1}{2}\bar{\omega} + 2w & 2w\bar{\omega} - w & \frac{1}{2} - w\omega & 0 & 0 & -w \end{bmatrix}$$

This is simple enough for the Maple function.


```
> jordan(ModButter, 'P');
```

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & -w & 0 & 0 & 0 & 0 \\ 0 & 0 & 2w & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -w \end{bmatrix}$$

```
> ModButterEigenvect := subs( { w = 1/16, cos(2*m*Pi/K) = c, sin(2*m*Pi/K) = s},
  map( simplify, map( evalc, subs( Buttervar, col(eval('P'),1)))));
```

$$ModButterEigenvect := \begin{bmatrix} 1, 2, -\frac{1}{3}c^2 + \frac{7}{6} + \frac{5}{6}c - \frac{1}{3}Ics + \frac{7}{6}Is, \frac{653}{216} - \frac{11}{108}c^2 + \frac{1}{216}c, \frac{35}{54}c - \frac{7}{27}c^2 + \frac{121}{54} + \frac{7}{6}Is - \frac{1}{3}Ics, \end{bmatrix}$$

$$\left. \frac{2}{27}c^3 + \frac{103}{54}c - \frac{14}{27}c^2 + \frac{7}{6} + \frac{2}{27}Is c^2 + \frac{121}{54}Is - \frac{14}{27}Ics \right]$$

 **Code generation**

Eigenstructure of the Kobbelt Scheme

Denis Zorin, Stanford University, February 1998

In this worksheet, we explore the eigenstructure of the subdivision matrix for Kobbelt's subdivision scheme. We compute guaranteed intervals for the magnitude of the largest eigenvalue, and formulas for computing corresponding eigenvector. The result is a file with three interval arithmetics C functions computing the magnitude of the largest eigenvalue for a large range of valences, and two functions to compute eigenvectors for the eigenvalue with the largest magnitude. In addition, we estimate the range of eigenvalues for large valences, which allows us to analyze C1-continuity for all valences.

✚ Utilities

▣ Subdivision matrix of the Kobbelt scheme

Define the blocks of the DFT-transformed subdivision matrix; perform some tests to check if the matrix was defined correctly. We use the following parameters: α and β are the coefficients of the 4-point scheme (we consider the case when the coefficients are 9/16 and

-1/16 respectively), $c = \cos\left(\frac{2\pi}{K}\right)$, and $\omega = e^{\left(\frac{2\pi}{K}\right)}$.

We test the correctness of the matrix in two ways: first, we compute a submatrix explicitly for the case $K = 4$, and check if the matrices agree; second, compute the eigenvectors and eigenvalues for $K=4$; in this case, the matrix has to have eigenvalue 1/2, and the corresponding complex eigenvector should be a part of a regular quadrilateral grid in the complex plane.

```
> Kobbelt := matrix([[alpha+4*beta*d-beta*(1+2*c),
4*d*beta^2/alpha-beta^2*(conjugate(omega)^2+2*c+1)/alpha, beta, 0, 0,
0, 0, 0, 0, 0, 0, 0],
[4*beta*alpha*d+alpha^2*(1+omega)-(1+omega)*alpha*beta,
4*beta^2*d-beta^2*(1+2*c)+2*alpha*beta*c+alpha^2,
(1+omega)*alpha*beta, beta^2*omega+alpha*beta, beta^2,
beta^2*conjugate(omega)+alpha*beta, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0], [alpha, alpha+beta*conjugate(omega), 0, 0, 0,
beta, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[alpha*omega, alpha+beta*omega, 0, beta, 0, 0, 0, 0, 0, 0, 0, 0],
[alpha, 0, alpha, 0, 0, 0, beta, 0, 0, 0, 0, 0],
[beta^2*conjugate(omega)+alpha^2+alpha*beta*omega,
alpha*beta*conjugate(omega)+alpha^2, beta^2*omega+alpha^2, alpha^2,
alpha*beta, alpha*beta*(1+conjugate(omega)), alpha*beta, alpha*beta,
beta^2, 0, 0, beta^2*conjugate(omega)], [beta*omega, alpha, 0, alpha,
0, 0, 0, beta, 0, 0, 0, 0], [(1+omega)*alpha*beta, alpha^2,
(1+omega)*alpha*beta, alpha^2, alpha^2, alpha^2, beta^2*(1+omega),
alpha*beta, alpha*beta, beta^2, alpha*beta, alpha*beta], [beta, alpha,
0, 0, 0, alpha, 0, 0, 0, 0, 0, beta],
[alpha*beta+alpha^2*omega+beta^2*omega^2, alpha^2+alpha*beta*omega,
beta^2+alpha^2*omega, (1+omega)*alpha*beta, alpha*beta, alpha^2,
alpha*beta*omega, beta^2*omega, 0, 0, beta^2, alpha*beta]]);
```

$$Kobbelt := \begin{bmatrix} \alpha+4\beta d-\beta(1+2c) & \frac{4d\beta^2}{\alpha}-\frac{\beta^2(\omega^2+2c+1)}{\alpha} & \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\alpha d+\alpha^2(1+\omega)-(1+\omega)\alpha\beta & 4\beta^2 d-\beta^2(1+2c)+2\alpha\beta c+\alpha^2 & (1+\omega)\alpha\beta & \beta^2\omega+\alpha\beta & \beta^2 & \beta^2\bar{\omega}+\alpha\beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha & \alpha+\beta\bar{\omega} & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha\omega & \alpha+\beta\omega & 0 & \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha & 0 & \alpha & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & 0 \\ \beta^2\bar{\omega}+\alpha^2+\alpha\beta\omega & \alpha\beta\bar{\omega}+\alpha^2 & \beta^2\omega+\alpha^2 & \alpha^2 & \alpha\beta & \alpha\beta(\bar{\omega}+1) & \alpha\beta & \alpha\beta & \beta^2 & 0 & 0 & \beta^2\bar{\omega} & \beta^2\bar{\omega} \\ \beta\omega & \alpha & 0 & \alpha & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 \\ (1+\omega)\alpha\beta & \alpha^2 & (1+\omega)\alpha\beta & \alpha^2 & \alpha^2 & \alpha^2 & \beta^2(1+\omega) & \alpha\beta & \alpha\beta & \beta^2 & \alpha\beta & \alpha\beta & \alpha\beta \\ \beta & \alpha & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & \beta \\ \alpha\beta+\alpha^2\omega+\beta^2\omega^2 & \alpha^2+\alpha\beta\omega & \beta^2+\alpha^2\omega & (1+\omega)\alpha\beta & \alpha\beta & \alpha^2 & \alpha\beta\omega & \beta^2\omega & 0 & 0 & \beta^2 & \alpha\beta & \alpha\beta \end{bmatrix}$$


```
> Kobconst := { alpha = 9/16, beta = -1/16 };
```

$$Kobconst := \left\{ \alpha = \frac{9}{16}, \beta = \frac{-1}{16} \right\}$$

```
> KobExpanded := subs(Kobconst, evalm(Kobbelt));
```

$$KobExpanded := \begin{bmatrix} \frac{5}{8}d - \frac{1}{4}c + \frac{1}{8} & \frac{1}{36}d - \frac{1}{144}\omega - \frac{1}{72}c - \frac{1}{144} & \frac{-1}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{9}{64}d + \frac{45}{128} + \frac{45}{128}\omega & \frac{1}{64}d + \frac{5}{16} - \frac{5}{64}c & -\frac{9}{256} - \frac{9}{256}\omega & \frac{1}{256}\omega - \frac{9}{256} & \frac{1}{256} & \frac{1}{256} & \frac{1}{256}\omega - \frac{9}{256} & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{16} & \frac{9}{16} - \frac{1}{16}\omega & 0 & 0 & 0 & \frac{-1}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{16}\omega & \frac{9}{16} - \frac{1}{16}\omega & 0 & \frac{-1}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{16} & 0 & \frac{9}{16} & 0 & 0 & 0 & \frac{-1}{16} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{256}\omega + \frac{81}{256} - \frac{9}{256}\omega & -\frac{9}{256}\omega + \frac{81}{256} & \frac{1}{256}\omega + \frac{81}{256} & \frac{81}{256} & \frac{-9}{256} & \frac{-9}{256}\omega - \frac{9}{256} & \frac{-9}{256} & \frac{-9}{256} & \frac{1}{256} & 0 & 0 & \frac{1}{256}\omega & 0 \\ -\frac{1}{16}\omega & \frac{9}{16} & 0 & \frac{9}{16} & 0 & 0 & 0 & \frac{-1}{16} & 0 & 0 & 0 & 0 & 0 \\ -\frac{9}{256} - \frac{9}{256}\omega & \frac{81}{256} & -\frac{9}{256} - \frac{9}{256}\omega & \frac{81}{256} & \frac{81}{256} & \frac{81}{256} & \frac{1}{256} + \frac{1}{256}\omega & \frac{-9}{256} & \frac{-9}{256} & \frac{1}{256} & \frac{-9}{256} & \frac{-9}{256} & 0 \\ \frac{-1}{16} & \frac{9}{16} & 0 & 0 & 0 & \frac{9}{16} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{16} \\ -\frac{9}{256} + \frac{81}{256}\omega + \frac{1}{256}\omega^2 & \frac{81}{256} - \frac{9}{256}\omega & \frac{1}{256} + \frac{81}{256}\omega & -\frac{9}{256} - \frac{9}{256}\omega & \frac{-9}{256} & \frac{81}{256} & -\frac{9}{256}\omega & \frac{1}{256}\omega & 0 & 0 & \frac{1}{256} & \frac{-9}{256} & \frac{-9}{256} \end{bmatrix}$$

Special case K = 4:

```
> KobRegular := map( unapply( subs( c = 0, x), x), map( simplify, map( evalc, subs( { d = 0, omega = I, c = 0 }, evalm(KobExpanded) ) ) ) );
```

Manually computed matrix for the regular case

```
> KobRegularManual := matrix( [[ 9/16 - I^2/16, 0, -1/16, 0, 0, 0], [ (81/256)*(I+1) - (9/256)*(I^2 - I), 81/256 + (1/256)*I^2, (-9/256)*(I+1), -9/256 + I/256, 1/256, -9/256 - (1/256)*I], [1, 0, 0, 0, 0, 0], [9/16, 9/16 + I/16, 0, 0, 0, -1/16], [0, 1, 0, 0, 0, 0], [9*I/16, 9/16 - I/16, 0, -1/16, 0, 0] ] );
```

$$KobRegularManual := \begin{bmatrix} \frac{5}{8} & 0 & \frac{-1}{16} & 0 & 0 & 0 \\ \frac{45}{128} + \frac{45}{128}I & \frac{5}{16} & -\frac{9}{256} - \frac{9}{256}I & -\frac{9}{256} + \frac{1}{256}I & \frac{1}{256} & -\frac{9}{256} - \frac{1}{256}I \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{16} & \frac{9}{16} + \frac{1}{16}I & 0 & 0 & 0 & \frac{-1}{16} \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{9}{16}I & \frac{9}{16} - \frac{1}{16}I & 0 & \frac{-1}{16} & 0 & 0 \end{bmatrix}$$

Check agreement with the regular case .

```
> norm( evalm(submatrix( KobRegular, 1..6, 1..6) - KobRegularManual) );
```

0

Check if the eigenvector for the eigenvalue 1/2 is a regular grid:

```
> eigenvects( KobRegular );
```

$$\left[\frac{1}{256}, 1, \{ [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0] \} \right], \left[\frac{-1}{128}, 1, \left\{ \left[0, 0, 0, 0, 0, 0, 0, \frac{1}{8}, 1, \frac{27}{8} - \frac{27}{8}I, -I, -\frac{1}{8}I \right] \right\} \right], \left[\frac{1}{2}, 1, \{ [1, 1+I, 2, 2+I, 2+2I, 1+2I, 3, 3+I, 3+2I, 3+3I, 2+3I, 1+3I] \} \right],$$

$$\left[\frac{-1}{32}, 1, \left\{ \left[0, 0, 0, 0, 0, 0, \frac{1}{2}I, \frac{3}{2} + \frac{3}{2}I, 1, \frac{1}{2} \right] \right\}, \left[\frac{-1}{16}, 1, \left\{ \left[0, 0, 0, 0, 0, 0, 1, 1, 1, 1 + I, I, I \right] \right\} \right], \right. \\ \left. \left[\frac{-1}{64}, 2, \left\{ \left[0, 0, 0, 0, 0, 0, -I, -4I, 9 - 9I, 4, 1 \right] \right\} \right], \right. \\ \left. \left[\frac{1}{32}, 2, \left\{ \left[0, 1, 0, 6 - 2I, 32, 6 + 2I, 0, 18 - 9I, 90 - 18I, 243, 90 + 18I, 18 + 9I \right] \right\} \right], \left[\frac{1}{8}, 3, \left\{ \right. \right. \\ \left. \left. \left[\frac{1}{8}, 0, 1, \frac{3}{4} - \frac{3}{8}I, 0, -\frac{3}{8} + \frac{3}{4}I, \frac{27}{8}, 3 - I, \frac{15}{8} - \frac{5}{4}I, 0, -\frac{5}{4} + \frac{15}{8}I, -1 + 3I \right], \right. \right. \\ \left. \left. \left[0, \frac{1}{8}, 0, \frac{3}{8} + \frac{1}{8}I, 1, \frac{3}{8} - \frac{1}{8}I, 0, \frac{3}{4} + \frac{3}{8}I, \frac{15}{8} + \frac{3}{8}I, \frac{27}{8}, \frac{15}{8} - \frac{3}{8}I, \frac{3}{4} - \frac{3}{8}I \right] \right\} \right]$$

Eigenvalues of the 0-th block for all valences:

```
> KobZeroBlock := map( unapply( subs( c = 1, 'x'), 'x'), map( simplify, map( evalc,
  subs( { d = 1, omega = 1, c = 1 }, evalm(KobExpanded) )))):
```

The largest eigenvalue is 1/4; we will see that the largest eigenvalue of one of the other blocks is always greater than 1/4, and the dominant eigenvalue is never in the 0-th block.

```
> eigenvals(KobZeroBlock);
```

$$\frac{1}{256}, \frac{-1}{32}, \frac{-1}{128}, \frac{-1}{16}, \frac{-1}{64}, \frac{-1}{64}, \frac{1}{4}, \frac{1}{4}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}$$

Characteristic polynomial of the subdivision matrix

Compute and factor the characteristic polynomial of the blocks of DFT-transformed subdivision matrix. The resulting polynomial is parameterized by $c = \cos\left(\frac{2\pi}{K}\right)$. The polynomial has a number of small roots, which do not depend on c , and a factor of degree 6. For illustrative purposes, and to guide us in the subsequent derivations, we compute all the roots of the polynomial numerically, and plot the magnitudes of the roots. Of course, neither the plot nor the computed values cannot be used in the analysis without additional verification.

We have already computed eigenvalues for the 0th block, and we can assume that $d = 0$

```
> KobCharpoly := subs( { s^3 = s*(1-c^2), s^2 = 1 - c^2, s^4 = (1-c^2)^2 }, factor(
  collect(charpoly( map( simplify, map( evalc, subs( s^2 = 1-c^2, map( simplify, subs( {
    d = 0, omega = c + I*s }, eval(KobExpanded) ))))), lambda), lambda))):
> KobCharpoly := factor( map( simplify, KobCharpoly));
```

$$KobCharpoly := -\frac{1}{72057594037927936} (32\lambda + 1)(128\lambda + 1)(1 + 64\lambda)^2(-1 + 42880\lambda^3 + 90\lambda + 576\lambda^2 c \\ + 9216\lambda^4 c + 49152\lambda^5 c - 18\lambda c - 5376\lambda^3 c + 448\lambda^3 c^2 + 983040\lambda^5 - 304128\lambda^4 - 2928\lambda^2 - 1048576\lambda^6)(256\lambda - 1) \\ (1 + 16\lambda)$$

```
> KobFactor6 := KobCharpoly/(
  (32*lambda+1)*(128*lambda+1)*(1+64*lambda)^2*(256*lambda-1)*(16*lambda+1)):
> KobFactor6 := collect(KobFactor6/lcoeff(KobFactor6, lambda), lambda);
```

$$KobFactor6 := \lambda^6 + \left(-\frac{3}{64}c - \frac{15}{16}\right)\lambda^5 + \left(-\frac{9}{1024}c + \frac{297}{1024}\right)\lambda^4 + \left(-\frac{335}{8192} - \frac{7}{16384}c^2 + \frac{21}{4096}c\right)\lambda^3 + \left(-\frac{9}{16384}c + \frac{183}{65536}\right)\lambda^2 \\ + \left(-\frac{45}{524288} + \frac{9}{524288}c\right)\lambda + \frac{1}{1048576}$$

Compute the eigenvalues; execution of this statement may take a while.

```
> EigenvalsList := seq([solve( subs( c = evalf( (n+1e-10)/100 ), KobFactor6 )], n =
  -100..100):
```

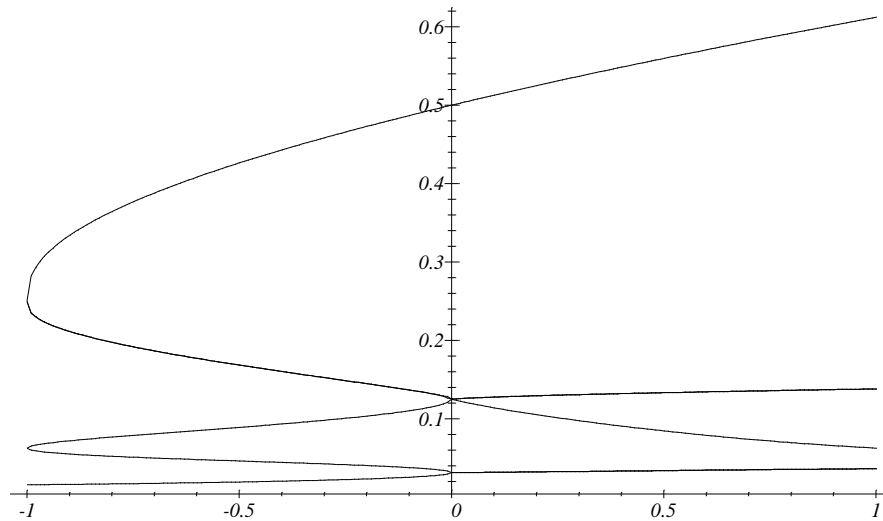
Convert the lists of eigenvalues to the form suitable for plotting

```
> EigenvalsPlotLists := seq( [ seq( [-1 + (i-1)/100, abs(op(j,
  op(i, [EigenvalsList])))], i = 1..201) ], j=1..6):
```

Plot of the magnitudes of eigenvalues as functions of c ; to see approximately the magnitudes of the eigenvalues for a block m of the subdivision matrix for valence K , draw a vertical line at $c = \cos\left(\frac{2\pi m}{K}\right)$ and find where it intersects the curves in the plot.

```
> display(seq(plot(op(i, [EigenvalsPlotLists]), color=black), i = 1..6), color=black,
```

```
axesfont=[TIMES,ITALIC,10], labels=['\','\']);
```



Eigenvalues

The roots of the characteristic polynomial of Kobbelt's scheme in general cannot be found explicitly. However, we can obtain enough information about the eigenvalues to verify C1-continuity. We prove that for any $m, k, m = 1 \dots k - 1$ the largest eigenvalue is real and unique, and that for $m \neq k - 1, 1$ the largest eigenvalue is less than the largest eigenvalue of blocks 1 and $k - 1$. We also show that the unique largest eigenvalue is a single eigenvalue in the interval $[0.5..0.613]$, for $k > 4$.

For the value $k = 3$, eigenvalues are examined separately. The proof is performed in several steps:

- (1). We show that for $c < 0$, all roots of the characteristic polynomial $P(c, \lambda)$ are less than 0.51 (actually, they are less than 0.5, but due to numerical nature of our calculations, we have to relax the upper boundary).
- (2). We show that for any $c = 0 \dots 1$, there is a unique real root μ in the interval $[0.5..0.613]$, and the function $\mu(c)$ is C1-continuous and increases.
- (3). We "deflate" the characteristic polynomial (that is, divide by the monomial $\lambda - \mu$) in symbolic form, with μ and c as indeterminates. Next, we verify that for all $\mu = .5 \dots .613$, and corresponding $c(\mu)$, that all roots of the deflated polynomial are inside the circle of radius 0.5 centered at 0 in the complex plane, that is, have magnitudes less than $\mu(c)$ for any $0 < c$. Using μ as the primary parameter is important, as c can be explicitly computed from μ , but not the other way.

As for $k > 4$, $.51 < \cos\left(\frac{2\pi}{k}\right)$ the largest eigenvalue cannot possibly correspond to a block m , for which $\cos\left(\frac{2m\pi}{k}\right) \leq 0$. From (3), it

follows that the largest root has to be the real root $\mu(c)$ for some c . As for any $1 < m, m < k - 1$, $\cos\left(\frac{2m\pi}{k}\right) < \cos\left(\frac{2\pi}{k}\right)$, and we

have shown (1) that $\mu(c)$, increases, and for any c $\mu(c)$ is the largest root, we conclude that the largest eigenvalue always corresponds to $m = 1$, is real, and is the unique eigenvalue in the range $0.5..0.613$.

On steps 1 and 3 we have to show that roots of a polynomial are inside a circle of radius r in the complex plane. This task is similar to the task of establishing

stability of a filter with the transfer function $\frac{1}{a(z)}$, where $a(z)$ is a polynomial. Such filter is stable, if all roots of the polynomial are inside the unit circle.

A variety of tests exist for this condition; for our purposes, the algebraic Marden-Jury test is convenient. With appropriate rescaling of the variable it can be used to prove that all roots of a polynomial are inside the circle of any given radius r . As the test requires only a simple algebraic calculation on the coefficients of the polynomial, it can be easily performed for symbolic and interval coefficients.

Finally, we compute the largest root of the characteristic polynomial numerically for all valences up to some maximum. For each computed root, we verify that the precision is at least $\epsilon = .1 \cdot 10^{-10}$: we use interval arithmetics to evaluate the polynomial at $\lambda_0 - \epsilon$ and $\lambda_0 + \epsilon$ and assert that the sign is guaranteed to change. There may be more than one root: we still have to prove that there is only a single root in the computed interval and that the rest of the roots are smaller. The maximal valence N is chosen in such a way that for

$N \cos\left(\frac{2\pi}{N}\right)$ is "sufficiently close" to 1. This means that for all $K > N$ corresponding eigenvalue differs from the limit value λ_∞ by no more than ε , where ε is small enough for us to establish, using interval arithmetics, that the Jacobian of the characteristic map is positive for eigenvectors computed using formulas derived below for all λ in the interval $[\lambda_\infty - \varepsilon, \lambda_\infty]$. The actual computation of the Jacobian and evaluation of the necessary contraction functions is performed in the C part of the code. We use maximal value 3000 here, just in case (below we see that 1450 is sufficient to require the interval for λ with the size of only $.1 \cdot 10^{-5}$).

▣ Marden-Jury test

MardenJury(a, var, rootrad) Compute Marden-Jury table for a polynomial **p** in variable **var**, with variable rescaled by **rootrad**.

Used to verify that all roots of a polynomial are inside the circle of radius **r**.

```
> MardenJury := proc(a::polynom, var::name, rootrad)
  local i, k, M, acol, tbl, restable;
  M := degree(a, var);
  for i from 0 to M do tbl[i, 0] := coeff(a, var, i)*rootrad^(i - M) od;
  for i to M do
    for k from 0 to M - i do tbl[i, k] := tbl[i - 1, 0]*tbl[i - 1, k] - tbl[i - 1, M - k - i + 1]*tbl[i - 1, M - i + 1] od
  od;
  for i to M do restable[i] := tbl[i, 0] od;
  eval(restable)
end
```

Interval version of Marden-Jury test

```
> IntervMardenJury := proc(a::polynom(interval), var::name, rootrad::numeric)
  local i, k, M, acol, tbl, restable;
  M := degree(a, var);
  for i from 0 to M do tbl[i, 0] := Interval_times(coeff(a, var, i), rootrad^(i - M)) od;
  for i to M do for k from 0 to M - i do tbl[i, k] := Interval_add(Interval_times(tbl[i - 1, 0], tbl[i - 1, k]),
    Interval_times(-1, Interval_times(tbl[i - 1, M - k - i + 1], tbl[i - 1, M - i + 1])))
  od
  od;
  for i to M do restable[i] := tbl[i, 0] od;
  eval(restable)
end
```

▣ Deflation

deflate(p, var, rootval) compute the coefficients of the polynomial $\frac{p(z)}{z - z_0}$; it is assumed that **p** is divisible by $z - z_0$.

var is the name of the variable, **rootval** is the root.

```
> deflate := proc(p::polynom, var::name, rootval)
  local i, dp, r;
  dp := 0;
  r := lcoeff(p, var);
  for i from degree(p, var) - 1 by -1 to 0 do dp := dp + r*var^i; r := coeff(p, var, i) + rootval*r od;
  dp
end
```

▣ Analysis of the eigenvalues

Now we perform steps 1-3 described above.

▣ (1). We show that for $c < 0$, all roots of the characteristic polynomial $P(c, \lambda)$ are less than 0.51 (actually, they are less than 0.5, but due to numerical nature of our calculations, we have to relax the upper boundary).

```
> MJtab := MardenJury(KobFactor6, lambda, 51/100);
> MJtabInterv := map(unapply('inapply'(dummy, c), dummy), MJtab);
> TestNegativeC := proc(cstart::numeric, cend::numeric, cstep::numeric)
  local MJ, cx;
```

```

global MJtabInterv;
  for cx from cstart by cstep to cend do
    MJ := map(unapply(dummy([ cx, cx + cstep ]), dummy), MJtabInterv);
    if 0 < op(2, MJ[1]) or op(1, MJ[2]) < 0 or op(1, MJ[3]) < 0 or op(1, MJ[4]) < 0 or op(1, MJ[5]) < 0 or
    op(1, MJ[6]) < 0 then ERROR('test failed for interval = ', [ cx, cx + cstep ])
    fi
  od;
  print('All tests passed')
end
>
Do tests, adjusting the step in c. This is not really necessary -- we can simply take the smallest step, but to save time we use
larger steps first.
TestNegativeC(-1.0, -0.65, 0.05);
All tests passed
> TestNegativeC(-0.6, -0.275, 0.025);
All tests passed
> TestNegativeC(-0.25, -0.06, 0.01);
All tests passed
> TestNegativeC(-0.05, 0., 0.005);
All tests passed

```

- ▣ (2). We show that for any $c = 0 \dots 1$, there is a unique real root μ in the interval $[0.5, 0.613]$, and the function $\mu(c)$ is C^1 -continuous and increases.

```

Solve the characteristic polynomial for c
> csolutions := [solve(KobFactor6, c)];
We are interested in the first solution only; we will verify later that the second one is out of the range [-1..1] for relevant
values of λ
> clambda := csolutions[1];
clambda :=  $\frac{1}{896} (-6144 \lambda^3 - 1920 \lambda^2 + 432 \lambda - 18$ 
 $+ 2 \sqrt{9437184 \lambda^6 + 13238272 \lambda^5 - 5451776 \lambda^4 + 393216 \lambda^3 + 30784 \lambda^2 - 3440 \lambda + 81} (-1 + 8 \lambda) / \lambda^2$ 
>
Compute the derivative.
> clambdadiff := simplify(diff(clambda, lambda));
The solution for  $\lambda = \frac{1}{2}$  can be computed explicitly.
> simplify(subs(lambda = 1/2, clambda));
0
The solution for  $\lambda = .613$  is outside the range.
> clambdaInterv := inapply(clambda, lambda): clambdaInterv(0.613);
[ 1.007236841, 1.007237163 ]
>
Show that the derivative is positive for  $\lambda$  in  $[0.5..0.613]$  (the upper bound is the upper estimate for  $\lambda(1)$ )
intervcdiff := inapply(clambdadiff, lambda):
> for xl from .5 by .004 to .613 do
  res := intervcdiff([xl, xl + .004]); if res_1 < 0 then ERROR('test failed for interval, [xl, xl + .004]) fi
od;
print(all tests passed)
all tests passed
We conclude that  $c_1(\lambda)$  increases from 0 to above 1 on  $[0.5..0.613]$ ; therefore, the inverse increases from 0.5 to approx.
0.613 on  $[0..1]$ .
The second solution is outside the range of c for this range of λ:
> inapply(csolutions[2], lambda)(0.5, 0.613);

```

[-28.12500027, -28.12499976]

We conclude that for $c > 0$ in the interval 0.5..0.613 there is a unique real solution.

- ☒ (3). We "deflate" the characteristic polynomial (that is, divide by the monomial $\lambda - \mu$) in the symbolic form, with μ and c as the indeterminates. Next, we verify that for all $c = 0 \dots 1$, and for all $\lambda = .5 \dots .613$, all roots of the deflated polynomial are inside the circle of radius 0.5 centered at 0 in the complex plane, that is, have magnitudes less than $\mu(c)$ for any $0 < c$.

Symbolic deflation; if we substitute a pair $c, \mu(c)$ we get the deflated polynomial for a specific value of c .

```
> deflatedKobFactor6 := collect( expand( deflate(KobFactor6, lambda, mu)),
lambda );
```

$$\begin{aligned} \text{deflatedKobFactor6} := & \lambda^5 + \left(-\frac{15}{16} - \frac{3}{64}c + \mu \right) \lambda^4 + \left(-\frac{9}{1024}c + \frac{297}{1024} - \frac{15}{16}\mu - \frac{3}{64}\mu c + \mu^2 \right) \lambda^3 \\ & + \left(\frac{21}{4096}c - \frac{335}{8192} - \frac{7}{16384}c^2 - \frac{9}{1024}\mu c + \frac{297}{1024}\mu - \frac{15}{16}\mu^2 - \frac{3}{64}\mu^2 c + \mu^3 \right) \lambda^2 \\ & + \left(\frac{183}{65536} - \frac{9}{16384}c + \frac{21}{4096}\mu c - \frac{335}{8192}\mu - \frac{7}{16384}\mu c^2 - \frac{9}{1024}\mu^2 c + \frac{297}{1024}\mu^2 - \frac{15}{16}\mu^3 - \frac{3}{64}\mu^3 c + \mu^4 \right) \lambda + \frac{9}{524288}c \\ & - \frac{45}{524288} - \frac{335}{8192}\mu^2 - \frac{9}{16384}\mu c - \frac{9}{1024}\mu^3 c + \frac{183}{65536}\mu + \frac{297}{1024}\mu^3 + \mu^5 + \frac{21}{4096}\mu^2 c - \frac{15}{16}\mu^4 - \frac{3}{64}\mu^4 c - \frac{7}{16384}\mu^2 c^2 \end{aligned}$$

Verify that for all c in 0..1 and μ in 0.5..0.613 deflated polynomial has roots of magnitude < 0.5

```
> TestDeflated := proc(lstart::numeric, lend::numeric, lstep::numeric)
local cf, i, MJ, lx, cinterv, deflatedinterv;
global deflatedMInterv, clambdaInterv;
for i from 0 to 5 do cf[i] := inapply(coeff(deflatedKobFactor6, lambda, i), c, mu) od;
for lx from lstart by lstep to lend do
    cinterv := clambdaInterv([lx, lx + lstep]);
    deflatedinterv := 0;
    for i from 0 to 4 do deflatedinterv := deflatedinterv + eval(cf[i](ciinterv, [lx, lx + lstep]))*lambda^i od;
    deflatedinterv := deflatedinterv + [1.0, 1.0]*lambda^5;
    MJ := IntervMardenJury(deflatedinterv, lambda, .5);
    if 0 < op(2, MJ[1]) or op(1, MJ[2]) < 0 or op(1, MJ[3]) < 0 or op(1, MJ[4]) < 0 or op(1, MJ[5]) < 0 or
    op(1, MJ[6]) < 0 then ERROR('test failed for interval = ', [lx, lx + lstep])
    fi
od;
print('All tests passed')
end
> TestDeflated(0.5, 0.613, 0.0005);
```

All tests passed

We conclude that in the range $c = 0 \dots 1$, all roots of the deflated polynomial have magnitudes less than 0.5

>

- ☒ Special case: $k = 3$

☒ Calculation of the largest eigenvalues with guaranteed precision

This function produces a table of approximate values of the eigenvalue with given precision for use with interval arithmetics in the C part of the analysis code; to avoid conversion problems, we write two integers: mantissa + exponent base 10. The last value is the limit value for infinity (computed with c set to 1 in the char. polynomial). The result is a C function written to a file; if the file name is 'default', then the output is written to the standard output.

The argument of the function is valence, the function returns the interval value for the largest eigenvalue. The body is just a large switch statement.

We assume that the exponents for eigenvalues are nonpositive, which is always the case for Kobbelt's scheme.

```
ComputeEigenvalues := proc(N::integer, eps::numeric, fname::string)
```

```
local K, intervKobFactor6, intervPi, interv, expandedKobFactor6, approxEV, r, deflatedKobFactor6, i, marTable, cK;
```

```
global KobFactor6;
```

```
Digits := 15;
```

```
intervKobFactor6 := inapply(KobFactor6, lambda, c);
```

```
intervPi := Interval_times([2.0, 2.0], Interval_arccos([0, 0]));
```

```

intervc := inapply(cos(2*intervPi / K), K);
fprintf(fname, 'virtual Float Eigenvalue(int K) {\n';
fprintf(fname, 'static INTEGER64 EV[] = {\n';
expandedKobFactor6 := subs(c = 1, KobFactor6);
approxEV := fsolve(expandedKobFactor6, λ, λ = .5 .. 1);
if 0 < op(2, intervKobFactor6(approxEV - eps, 1)) or op(1, intervKobFactor6(approxEV + eps, 1)) < 0 then
  ERROR('fsolve precision failure for infinity')
fi;
fprintf(fname, 'CONST64(%d),CONST64(%d),CONST64(%d),\n', op(1, approxEV - eps), op(1, approxEV + eps),
  10^(-op(2, approxEV)));
fprintf(fname, 'CONST64(0),CONST64(0),CONST64(0),CONST64(0),CONST64(0),CONST64(0),\n');
for K from 3 to N do
  if (K - 3) mod 100 = 0 then print(K) fi;
  cK := intervc(K);
  expandedKobFactor6 := subs(c = cos(2*π / K), KobFactor6);
  approxEV := fsolve(expandedKobFactor6, λ, λ = .25 .. 1);
  if 0 < op(2, intervKobFactor6(approxEV - eps, cK)) or op(1, intervKobFactor6(approxEV + eps, cK)) < 0 then
    ERROR('fsolve precision failure for K = ', K)
  fi;
  fprintf(fname, 'CONST64(%d),CONST64(%d),CONST64(%d),\n', op(1, eval(approxEV - eps)),
    op(1, eval(approxEV + eps)), 10^(-op(2, approxEV)))
od;
fprintf(fname, 'CONST64(0);\n return Float(EV[3*K],EV[3*K+1])/Float(EV[3*K+2]);\n}\n');
NULL
end

```

▣ The derivative of the largest eigenvalue with respect to c at infinity.

To establish C1-continuity for all valences, we need to analyze behavior of the magnitude of the largest eigenvalue as the function of the valence, as the valence increases to infinity (c approaches 1). We estimate a constant B , such that $|\lambda - \lambda_\infty| < B|c - 1|$, sufficiently close to 1. This constant can be taken to be the maximum of $\left| \frac{\partial}{\partial c} \lambda \right|$, or, equivalently, as maximum of $\left| \frac{\partial}{\partial \lambda} c \right|^{(-1)}$; as the characteristic polynomial is quadratic in c , the latter is relatively easy to compute. Once B is known, we can estimate the size ε of the interval for λ near λ_∞ , such that if the characteristic map is injective and regular for all these values, it is sufficient to

establish C1-continuity for $K_0 < K$, where $\frac{\varepsilon}{B} < 1 - \cos\left(\frac{2\pi}{K_0}\right)$

```

> intervcdiff := inapply( clambdadiff, lambda):
Evaluate for all lambda in the range 0.7..1; step 0.001 gives reasonable bounds; this may take some time.
> cdiffinterv := [];
for i from 0 to 299 do
  cdiffinterv := Interval_union(cdiffinterv, intervcdiff([0.7+i*0.001,
0.7+(i+1)*0.001]));
od: eval(cdiffinterv);

```

```

cdiffinterv := [ ]
[9.984080932, 15.63504120]

```

```

> B := op(2, Interval_reciprocal( op(1, cdiffinterv)));
B := .1001594446

```

For example, if we use the interval of size $.1 \cdot 10^{-5}$ for λ_∞ , we have to consider all valences up to the valence for which

$B \left| \cos\left(\frac{2\pi}{K}\right) - 1 \right| < .1 \cdot 10^{-5}$, which turns out to be approx. 1450.

```

> Interval_times( B, Interval_add( Interval_cos( Interval_times( Interval_times( 2,
2*Interval_arccos(0.0)), Interval_reciprocal(1450))),-1));
[-.9403469458 10-6, -.9403269135 10-6]

```

▣ Eigenvectors

Finally, we derive the expressions for the complex eigenvector of the largest eigenvalue of the subdivision matrix. We use the fact that the largest eigenvalue has multiplicity 1 and is a real eigenvalue of the first subblock $B_{0,0}$ of the subdivision matrix.

First part of the eigenvector

```
> KobBlock00 := subs( s^2 = 1-c^2, map( evalc, subs( { d = 0, omega = c + I*s},
submatrix( KobExpanded, 1..6,1..6))));
```

KobBlock00 :=

$$\begin{bmatrix} \frac{5}{8} + \frac{1}{8}c & -\frac{1}{72}c^2 - \frac{1}{72}c + \frac{1}{72}Ics & -\frac{1}{16} & 0 & 0 & 0 \\ \frac{45}{128} + \frac{45}{128}c + \frac{45}{128}Is & \frac{5}{16} - \frac{5}{64}c & -\frac{9}{256} - \frac{9}{256}c - \frac{9}{256}Is & \frac{1}{256}c + \frac{1}{256}Is - \frac{9}{256} & \frac{1}{256} & \frac{1}{256}c - \frac{9}{256} - \frac{1}{256}Is \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{16} & \frac{9}{16} - \frac{1}{16}c + \frac{1}{16}Is & 0 & 0 & 0 & -\frac{1}{16} \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{9}{16}c + \frac{9}{16}Is & \frac{9}{16} - \frac{1}{16}c - \frac{1}{16}Is & 0 & -\frac{1}{16} & 0 & 0 \end{bmatrix}$$

The characteristic polynomial of this submatrix is exactly the degree 6 factor of the characteristic polynomial of the whole matrix:

```
> collect( simplify( subs( { s^3 = s*(1-c^2), s^2 = 1 - c^2},
charpoly(KobBlock00,lambda))), lambda) - KobFactor6;
```

0

```
> redBlock00 := submatrix( evalm( KobBlock00 - lambda * &*()), 2..6, 1..6):
> v0 := map( simplify, subs( { s^3 = (1-c^2)*s, s^2 = 1-c^2, _t[1] = 1}, linsolve(
redBlock00, vector([0,0,0,0,0]) )));
```

$$v0 := \left[\lambda, 9 \frac{\lambda(2c^2\lambda - 1 - c + 384\lambda^2 - Is + 384c\lambda^2 - 2560c\lambda^3 - 2560\lambda^3 + 2I\lambda cs - 2560I\lambda^3s + 384I\lambda^2s + 2c\lambda)}{\%1}, \right. \\ \left. 1, 9 \frac{I\lambda(-9s + 9Ic - 2Ic^2 + 2sc - 228I\lambda c + 1344Ic\lambda^2 + 240s\lambda - 144I\lambda + 4096I\lambda^3 + 10I - 1344\lambda^2s)}{\%1}, \right. \\ \left. 9 \frac{2c^2\lambda - 1 - c + 384\lambda^2 - Is + 384c\lambda^2 - 2560c\lambda^3 - 2560\lambda^3 + 2I\lambda cs - 2560I\lambda^3s + 384I\lambda^2s + 2c\lambda}{\%1}, \right. \\ \left. -9 \frac{I\lambda(-8Ic - 4096Ic\lambda^3 + 10s - 9I - 144s\lambda + 144I\lambda c + 12s\lambda c + 4096\lambda^3s + 240I\lambda - 1344I\lambda^2 - 12I\lambda c^2)}{\%1} \right]$$

$$\%1 := 80\lambda + 4c^2\lambda - 16c\lambda - 2112\lambda^2 - 5120c\lambda^3 - 1 + 576c\lambda^2 - 65536\lambda^4 + 20480\lambda^3$$

Verify agreement with the regular case:

```
> subs( { s = 1, c = 0, lambda = 1/2}, eval(v0) );
```

$$\left[\frac{1}{2}, \frac{1}{2} + \frac{1}{2}I, 1, 1 + \frac{1}{2}I, 1 + I, \frac{1}{2} + I \right]$$

Second part, separate real and imaginary parts

Now we compute the second part of the vector:

```
> KobBlock10 := submatrix( KobExpanded, 7..12, 1..6);
```



```

KobBlock10 :=

$$\begin{bmatrix} \frac{9}{16} & 0 & \frac{9}{16} & 0 & 0 & 0 \\ \frac{1}{256}\omega + \frac{81}{256} - \frac{9}{256}\omega & -\frac{9}{256}\omega + \frac{81}{256} & \frac{1}{256}\omega + \frac{81}{256} & \frac{81}{256} & \frac{-9}{256} & -\frac{9}{256} - \frac{9}{256}\omega \\ -\frac{1}{16}\omega & \frac{9}{16} & 0 & \frac{9}{16} & 0 & 0 \\ -\frac{9}{256} - \frac{9}{256}\omega & \frac{81}{256} & -\frac{9}{256} - \frac{9}{256}\omega & \frac{81}{256} & \frac{81}{256} & \frac{81}{256} \\ \frac{-1}{16} & \frac{9}{16} & 0 & 0 & 0 & \frac{9}{16} \\ -\frac{9}{256} + \frac{81}{256}\omega + \frac{1}{256}\omega^2 & \frac{81}{256} - \frac{9}{256}\omega & \frac{1}{256} + \frac{81}{256}\omega & -\frac{9}{256} - \frac{9}{256}\omega & \frac{-9}{256} & \frac{81}{256} \end{bmatrix}$$

> KobBlock11lambda := submatrix ( evalm( KobExpanded - lambda * &*( ), 7..12,
7..12);

```

$$KobBlock11lambda := \begin{bmatrix} -\frac{1}{16} - \lambda & 0 & 0 & 0 & 0 & 0 \\ \frac{-9}{256} & -\frac{9}{256} - \lambda & \frac{1}{256} & 0 & 0 & \frac{1}{256}\omega \\ 0 & \frac{-1}{16} & -\lambda & 0 & 0 & 0 \\ \frac{1}{256} + \frac{1}{256}\omega & \frac{-9}{256} & \frac{-9}{256} & \frac{1}{256} - \lambda & \frac{-9}{256} & \frac{-9}{256} \\ 0 & 0 & 0 & 0 & -\lambda & \frac{-1}{16} \\ -\frac{9}{256}\omega & \frac{1}{256}\omega & 0 & 0 & \frac{1}{256} & -\frac{9}{256} - \lambda \end{bmatrix}$$

```

> v1 := map(simplify, subs( { s^4 = (1-c^2)^2, s^2 = 1-c^2, s^3 = (1-c^2)*s}, map(
simplify, map( evalc, subs( omega = I*s + c, evalm( - inverse(KobBlock11lambda) &*
KobBlock10 &* v0)))))):

```

Put together the vector:

```

> KobEigenvect := vector( [seq( v0[i], i = 1..6), seq( v1[i], i = 1..6) ]):

```

Verify agreement with the regular case:

```

> subs( { s = 1, c = 0, lambda = 1/2}, map( simplify, evalm( eval(KobEigenvect) ) )
);

```

$$\left[\frac{1}{2}, \frac{1}{2} + \frac{1}{2}I, 1, 1 + \frac{1}{2}I, 1 + I, \frac{1}{2} + I, \frac{3}{2}, \frac{3}{2} + \frac{1}{2}I, \frac{3}{2} + I, \frac{3}{2} + \frac{3}{2}I, 1 + \frac{3}{2}I, \frac{1}{2} + \frac{3}{2}I \right]$$

Separate real and complex parts

```

> KobEigenvectRe := map( evalc, map( Re, KobEigenvect)):

```

In addition, scale imaginary part by 1/s

```

> KobEigenvectIm := map( simplify, evalm( (1/s) * map( evalc, map( Im,
KobEigenvect))))):

```

⊕ Code generation

Eigenstructure of the Loop Scheme

Denis Zorin, January 1998

In this worksheet we compute the eigenvalues and eigenvectors of the subdivision matrix of the Loop scheme. This analysis repeats the original derivation of Loop from his MS thesis. We also compute eigenvalues and eigenvectors of the modified Loop scheme, which always has largest eigenvalue equal to 1/2.

+ Utilities

- Loop scheme subdivision matrix, eigenvalues, eigenvectors

```
> assume( c, real); additionally( c >= -1 ); additionally( c <= 1);
> Loop := matrix( [[ (1 - K*alpha)*d,      K*alpha*d,      0,0],
                  [      (3/8)*d,      3/8 + (1/4)*c,      0,0],
                  [      (1/16)*d,      5/8 + (1/8)*c, 1/16,
(1/16)*(1+conjugate(omega))],
                  [      (1/8)*d, (3/8)*(1 + omega), 0, 1/8]
                  ]);
```

$$Loop := \begin{bmatrix} (1-K\alpha)d & K\alpha d & 0 & 0 \\ \frac{3}{8}d & \frac{3}{8} + \frac{1}{4}c & 0 & 0 \\ \frac{1}{16}d & \frac{5}{8} + \frac{1}{8}c & \frac{1}{16} & \frac{1}{16} + \frac{1}{16}\omega \\ \frac{1}{8}d & \frac{3}{8} + \frac{3}{8}\omega & 0 & \frac{1}{8} \end{bmatrix}$$

```
> Loopvar := { omega = c + I*s};
```

Loopvar := { ω = c + I s }

```
> evalues := eigenvals(subs( { d = 0, op(Loopvar)}, evalm(Loop)));
```

$$evalues := 0, \frac{3}{8} + \frac{1}{4}c, \frac{1}{16}, \frac{1}{8}$$

```
> EV := max(evalues);
```

$$EV := \frac{3}{8} + \frac{1}{4}c$$

```
> LoopZero := map( evalc, subs( { d = 1, c = 1, omega = 1}, evalm(Loop)));
```

```
> eigenvals( LoopZero);
```

$$1, \frac{1}{8}, \frac{1}{16}, \frac{5}{8} - K\alpha$$

Minimal value of the largest eigenvalue of the first block is 1/4; determine the range for α ; α clearly should be less than 5/8K and greater than the following number:

```
> AlphaCrit := solve( 5/8 - K*alpha = 3/8 + 1/4*cos(2*Pi/K), alpha );
```

$$AlphaCrit := -\frac{1}{4} \frac{-1 + \cos\left(2 \frac{\pi}{K}\right)}{K}$$

```
> evecs := subs( s^2 = 1-c^2, [eigenvects(map(evalc,subs( { d = 0, op(Loopvar)},
evalm(Loop))))]);
```

$$evecs := \left[\left[\frac{3}{8} + \frac{1}{4}c, 1, \left[0, 1, \frac{1}{2} \frac{30c + 26 + 4c^2}{4c^2 + 9c + 5}, \frac{3}{2} \frac{I(-Ic + s - I)}{c + 1} \right] \right], \left[\frac{1}{8}, 1, \{ [0, 0, -I(I + Ic + s), 1] \} \right], \left[\frac{1}{16}, 1, \{ [0, 0, 1, 0] \} \right], [0, 1, \{ [1, 0, 0, 0] \}] \right]$$

```

> for i from 1 to vectdim(evects) do
  if op(1,op(i, evects)) = EV then v := op(1, op(3, op(i, evects))); fi;
  od;
> LoopEigenvector := map(simplify, map(evalc, eval(v)));

```

$$\text{LoopEigenvector} := \left[0, 1, \frac{2c+13}{5+4c}, \frac{3c+1+Is}{2c+1} \right]$$

Drop the first element

```

> LoopEigenvector := vector( [ seq(LoopEigenvector[i], i = 2..4)]);

```

$$\text{LoopEigenvector} := \left[1, \frac{2c+13}{5+4c}, \frac{3c+1+Is}{2c+1} \right]$$

+ Generate code

Modified Loop scheme subdivision matrix, eigenvalues, eigenvectors

This is the expression for all blocks except 0th; the 0th block is the same as for the standard Loop.

```

> ModLoop := matrix( [[ (1 - K*alpha)*d,      K*alpha*d,      0,0],
  [          (3/8)*d,      (1/2)^m,      0,0],
  [          (1/16)*d,      5/8 + (1/8)*c, 1/16,
  (1/16)*(1+conjugate(omega))],
  [          (1/8)*d, (3/8)*(1 + omega), 0, 1/8]
  1]);

```

$$\text{ModLoop} := \begin{bmatrix} (1-K\alpha)d & K\alpha d & 0 & 0 \\ \frac{3}{8}d & \left(\frac{1}{2}\right)^m & 0 & 0 \\ \frac{1}{16}d & \frac{5}{8} + \frac{1}{8}c & \frac{1}{16} & \frac{1}{16} + \frac{1}{16}\omega \\ \frac{1}{8}d & \frac{3}{8} + \frac{3}{8}\omega & 0 & \frac{1}{8} \end{bmatrix}$$

```

> mevalues := eigenvals(subs( { d = 0, op(Loopvar)}, evalm(ModLoop)));

```

$$\text{mevalues} := 0, \left(\frac{1}{2}\right)^m, \frac{1}{16}, \frac{1}{8}$$

The admissible range of α is obvious.

```

> mevects := subs( s^2 = 1-c^2, [eigenvects(map(evalc, subs( { d = 0, op(Loopvar) , m =
  1}, evalm(ModLoop))))]);

```

$$\text{mevects} := \left[\begin{array}{l} [0, 1, \{[1, 0, 0, 0]\}], \left[\frac{1}{16}, 1, \{[0, 0, 1, 0]\}\right], \left[\frac{1}{2}, 1, \left\{0, 1, \frac{12}{7} + \frac{4}{7}c, I(-Ic + s - I)\right\}\right], \left[\frac{1}{8}, 1, \{[0, 0, -I(I+Ic+s), 1]\}\right] \end{array} \right]$$

```

> for i from 1 to vectdim(mevects) do
  if op(1,op(i, mevects)) = 1/2 then mv := op(1, op(3, op(i, mevects))); fi;
  od;
> ModLoopEigenvector := map( evalc, map(simplify, mv));

```

$$\text{ModLoopEigenvector} := \left[0, 1, \frac{12}{7} + \frac{4}{7}c, c+1+Is \right]$$

Drop the first element

```

> ModLoopEigenvector := vector( [ seq(ModLoopEigenvector[i], i = 2..4)]);

```

$$\text{ModLoopEigenvector} := \left[1, \frac{12}{7} + \frac{4}{7}c, c + 1 + Is \right]$$

 **Generate code**

Eigenstructure of the Catmull-Clark scheme

Denis Zorin, February 1998

In this worksheet, we examine the eigenstructure of the subdivision matrices of the Catmull-Clark subdivision scheme. This scheme was analyzed in the papers by Ball and Storry (tangent plane continuity) and Peters and Reif (C1-continuity). We mostly follow the analysis found in the latter paper. In addition, we determine the range of coefficients for the extraordinary vertex, for which the scheme is C1-continuous.

⊕ Utilities

▣ Subdivision matrix

The constants following Peter and Reif, with K replacing n.

```
> assume( c, real); additionally( c <= 1 ); additionally( c >= -1 );
CCconst := { p1 = 1/64, p2 = 3/32, p3 = 9/16, q1 = 1/16, q2 = 3/8, r = 1/4 };
CCvar := { omega = exp( 2*Pi*I*m/K), c = cos( 2*Pi*m/K) };
```

$$CCconst := \{ p1 = \frac{1}{64}, p2 = \frac{3}{32}, p3 = \frac{9}{16}, q1 = \frac{1}{16}, q2 = \frac{3}{8}, r = \frac{1}{4} \}$$

$$CCvar := \{ \omega = e^{\left(\frac{2\pi m}{K} \right)}, c = \cos\left(2 \frac{\pi m}{K} \right) \}$$

The DFT-transformed subdivision matrix; the order of rows is different from P. & R. (there is a typo in Peters and Reif in row 4, elements 1 and 3).

```
> CC := matrix( [ [alpha*d,      beta*d,      gamma*d,0,0,0,0],
                  [ q2*d, 2*q1*c+q2, q1*(1+conjugate(omega)),0,0,0,0],
                  [  r*d, r*(1+omega),      r,0,0,0,0],
                  [ p2*d, 2*p1*c+p3, p2*(1+conjugate(omega)), p2,
p1,0,p1*conjugate(omega)],
                  [ q1*d, q1*omega + q2,      q2,      q1,
q1,0,0],
                  [ p1*d, p2*(1+omega),      p3, p1*(1+omega), p2,
p1, p2 ],
                  [ q1*d, q1 + q2*omega, q2      ,q1*omega,0,0,q1]
1]);
```

$$CC := \begin{bmatrix} \alpha d & \beta d & \gamma d & 0 & 0 & 0 & 0 \\ q2 d & 2 q1 c + q2 & q1 (1 + \omega) & 0 & 0 & 0 & 0 \\ r d & r (1 + \omega) & r & 0 & 0 & 0 & 0 \\ p2 d & 2 p1 c + p3 & p2 (1 + \omega) & p2 & p1 & 0 & p1 \omega \\ q1 d & q1 \omega + q2 & q2 & q1 & q1 & 0 & 0 \\ p1 d & p2 (1 + \omega) & p3 & p1 (1 + \omega) & p2 & p1 & p2 \\ q1 d & q1 + q2 \omega & q2 & q1 \omega & 0 & 0 & q1 \end{bmatrix}$$

```
> CCExpanded := map( evalc, subs( { omega = c + s*I, op(CCconst)}, eval(CC)));
```

We are primarily interested in m = 1; in this case d = 0 and we ignore the first row and column

```
> A00 := submatrix( CCExpanded, 2..3,2..3);
```

$$A00 := \begin{bmatrix} \frac{1}{8}c + \frac{3}{8} & \frac{1}{16} + \frac{1}{16}c - \frac{1}{16}Is \\ \frac{1}{4} + \frac{1}{4}c + \frac{1}{4}Is & \frac{1}{4} \end{bmatrix}$$

```
> A10 := submatrix( CCExpanded, 4..7,2..3):
```

```
> A11 := submatrix( CCExpanded, 4..7, 4..7):
```

```
> CCZero := map( evalc, subs( { gamma = 1 - alpha - beta, c = 1, omega = 1, d = 1,
op(CCconst)}, eval(CC)));
```

$$CCZero := \begin{bmatrix} \alpha & \beta & 1-\alpha-\beta & 0 & 0 & 0 & 0 \\ \frac{3}{8} & \frac{1}{2} & \frac{1}{8} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{3}{32} & \frac{19}{32} & \frac{3}{16} & \frac{3}{32} & \frac{1}{64} & 0 & \frac{1}{64} \\ \frac{1}{16} & \frac{7}{16} & \frac{3}{8} & \frac{1}{16} & \frac{1}{16} & 0 & 0 \\ \frac{1}{64} & \frac{3}{16} & \frac{9}{16} & \frac{1}{32} & \frac{3}{32} & \frac{1}{64} & \frac{3}{32} \\ \frac{1}{16} & \frac{7}{16} & \frac{3}{8} & \frac{1}{16} & 0 & 0 & \frac{1}{16} \end{bmatrix}$$

▣ Eigenvalues

The eigenvalues of the matrix are eigenvalues of $A_{0,0}$ and of $A_{1,1}$.

All eigenvalues of $A_{1,1}$ do not depend on c and are less than $1/8$:

```
> map( simplify, subs( s^2 = 1-c^2, [eigenvals( A11)]));
```

$$\left[\frac{1}{64}, \frac{1}{16}, \frac{1}{8}, \frac{1}{32} \right]$$

It can be seen that whenever the eigenvalues are real, one of them is greater than $1/4$.

```
> eigenv := map( evalc, subs( { s^2 = 1 - c^2 } , [eigenvals(A00)]));
```

$$eigenv := \left[\frac{1}{16}c + \frac{5}{16} + \frac{1}{16}\sqrt{c^2 + 10c + 9}, \frac{1}{16}c + \frac{5}{16} - \frac{1}{16}\sqrt{c^2 + 10c + 9} \right]$$

Make sure that the roots are never equal; then the largest is determined simply by comparing them for any value of c .

```
> solve( (op(1, eigenv) - op(2, eigenv))^2 = 0);
```

-9, -1

Figure out which one is larger.

```
> if simplify( subs( c = 0, op(1, eigenv) > op(2, eigenv))) then
```

```
    lambda1 := op(1, eigenv);
```

```
else
```

```
    lambda1 := op(2, eigenv);
```

```
fi;
```

$$\lambda_1 := \frac{5}{16} + \frac{1}{16}c + \frac{1}{16}\sqrt{9 + 10c + c^2}$$

The larger eigenvalue increases with c on the interval of interest. We conclude that the largest eigenvalue for $K > 4$ is guaranteed to be in the

1st or 0th block of the subdivision matrix. This is also true for $K = 3$, because there are only two blocks, excluding 0th, and they are complex-conjugate.

```
> solve( {diff(lambda1, c) > 0, abs(c) < 1});
```

{-1 < c, c < 1}

The largest eigenvalues of the 0th block:

```
> evZero := eigenvals( submatrix( CCZero, 1..3, 1..3) );
```

$$evZero := 1, \frac{1}{2}\alpha - \frac{1}{8} + \frac{1}{8}\sqrt{16\alpha^2 - 8\alpha - 3 + 8\beta}, \frac{1}{2}\alpha - \frac{1}{8} - \frac{1}{8}\sqrt{16\alpha^2 - 8\alpha - 3 + 8\beta}$$

Determine the ranges for α and β

Determine when the eigenvalues are real.

```
> solve( (4*( op(2, [evZero]) - op(3, [evZero] )))^2 >= 0 );
```

$$\left\{ \frac{3}{8} - 2\alpha^2 + \alpha \leq \beta \right\}$$

Determine where the eigenvalues intersect the level $\lambda_1\left(-\frac{1}{2}\right)$ which is the minimal value of the eigenvalue of the first block, achieved for $K = 3$.

```
> solve( abs(op(2, [evZero])) = subs( c = -1/2, lambda1 ) );
{ alpha = alpha, beta = -9/4*alpha + 117/64 - 1/4*alpha*sqrt(17) + 13/64*sqrt(17) }, { beta = 9/4*alpha + 45/64 + 1/4*alpha*sqrt(17) + 5/64*sqrt(17), alpha = alpha }
{ beta = -9/4*alpha + 117/64 - 1/4*alpha*sqrt(17) + 13/64*sqrt(17), alpha = alpha }, { beta = 9/4*alpha + 45/64 + 1/4*alpha*sqrt(17) + 5/64*sqrt(17), alpha = alpha }
> solve( abs( op(3, [evZero])) = subs( c = -1/2, lambda1 ) );
{ beta = -9/4*alpha + 117/64 - 1/4*alpha*sqrt(17) + 13/64*sqrt(17), alpha = alpha }, { beta = 9/4*alpha + 45/64 + 1/4*alpha*sqrt(17) + 5/64*sqrt(17), alpha = alpha }
```

The parabola is the boundary of the region where the eigenvalues are complex.

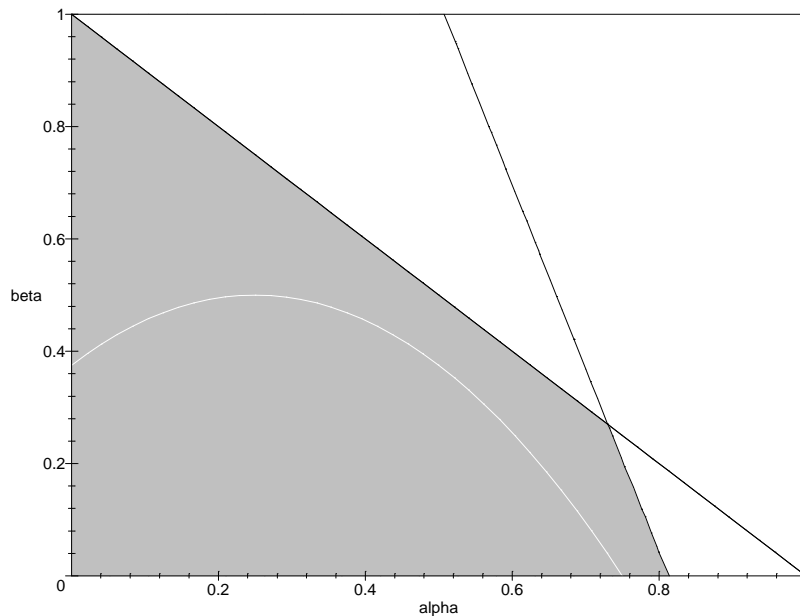
```
> solve( ( op(2, [evZero]))^2 - (op(3, [evZero]))^2 = 0 );
{ beta = 3/8 - 2*alpha^2 + alpha, alpha = alpha }, { alpha = 1/4, beta = beta }
```

The following plot shows the region in the $\alpha - \beta$ plane where the eigenvalues of 0th block are less than $\lambda_1\left(-\frac{1}{2}\right)$. For the coefficients α, β, γ to be positive, α and β have to be inside the triangle $[0,0], [1,0], [0,1]$.

For the magnitudes of eigenvalues to be less than $\lambda_1\left(-\frac{1}{2}\right)$ they have to be in the grey region, the equation of the line delimiting the region is

$$\beta = -\frac{9\alpha}{4} + \frac{117}{64} - \frac{1}{4}\alpha\sqrt{17} + \frac{13}{64}\sqrt{17}$$

```
> with(plots): display( PLOT(POLYGONS([[1,0], [1,1], [0,1]], COLOR(RGB, 1.0,1.0,1.0), STYLE(PATCH))), contourplot( max(abs(op(3, [evZero])), abs(op(2, [evZero]))) , alpha = 0..1, beta = 0..1, grid = [20,20], contours = [subs(c = -1/2, lambda1)], filled=true, coloring=[grey, white]), plot( 3/8 - 2*alpha^2 + alpha, alpha = 0..1, 0..1, color = white), plot( 1-alpha, alpha = 0..1, color = black) );
```



Eigenvalue summary. Whenever the coefficients α and β are in the region depicted above, the largest eigenvalue of the subdivision matrix is the eigenvalue of the first block, and it is greater than any other eigenvalue of the subdivision matrix.

[-] Eigenvector

The eigenvector of $A_{0,0}$ can be immediately seen from the matrix.

```
> v0 := [ 4*lambda - 1, (1+ c + I*s) ];
```

$$v0 := [4\lambda - 1, 1 + c + Is]$$

Invert $A_{1,1} - \lambda I$

```
> Allinv := map( simplify, subs( s^2 = 1 - c^2, eval(map( expand, evalm(inverse(
  evalm(A11 - lambda*&*()))))))):
```

compute v_1 as $-(A_{1,1} - \lambda I)A_{1,0}v_0$

```
> v1 := map( simplify, subs( { s^3 = s*(1-c^2), s^2 = 1-c^2}, map( expand, evalm(
  -Allinv &* A10 &* v0 )))):
```

We do not substitute the value for $\det A_{1,1} - \lambda I$

```
> CCEigenvector := vector( [ v0[1], v0[2], v1[1], v1[2], v1[3], v1[4] ] );
```

$$CCEigenvector := \left[4\lambda - 1, 1 + c + Is, 2 \frac{-60\lambda + 3 + 16c\lambda^2 + 20c\lambda + 288\lambda^2}{256\lambda^2 - 40\lambda + 1}, 2 \left(\frac{1024c\lambda^3 + 1024Is\lambda^3 + 6144\lambda^3 + 1152c\lambda^2 + 1120Is\lambda^2 - 384\lambda^2 - 156c\lambda - 96\lambda - 196Is\lambda + 6 + 5c + 5Is}{(256\lambda^2 - 40\lambda + 1)(-1 + 16\lambda)}, 49152\lambda^4 + 49152c\lambda^4 + 49152Is\lambda^4 + 77056c\lambda^3 + 256c^2\lambda^3 + 76800Is\lambda^3 + 76800\lambda^3 + 256Is\lambda^3c + 400c^2\lambda^2 + 400Is\lambda^2c - 11808\lambda^2 - 11808Is\lambda^2 - 11408c\lambda^2 + 100c^2\lambda + 60\lambda + 100Is\lambda c + 160c\lambda + 60Is\lambda + 15 + 15c + 15Is \right) / \left((16384\lambda^3 - 2816\lambda^2 + 104\lambda - 1)(-1 + 16\lambda), (1024\lambda^3 + 6144Is\lambda^3 + 6144c\lambda^3 - 384Is\lambda^2 + 32c^2\lambda^2 - 384c\lambda^2 + 1120\lambda^2 + 32Is\lambda^2c - 96c\lambda - 96Is\lambda + 40Is\lambda c + 40c^2\lambda - 196\lambda + 5 + 6Is + 6c) / (-1 + 56\lambda - 896\lambda^2 + 4096\lambda^3) \right) \right]$$

Verify that the vector is correct in the regular case.

```
> map( simplify, subs( { lambda = 1/2, s = 1, c = 0}, eval(CCEigenvector)));
```

$$[1, 1 + I, 2, 2 + I, 2 + 2I, 1 + 2I]$$

[+] Code generation