This worksheet contains the symbolic part of the analysis of tangent 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 $K > 8$, 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 $(c, \text{real})$; assume $(K, \text{integer})$; additionally $(K >= 3)$.
- Butterfly := matrix([[(1/2)*omega + 2*w, 2*w*omega - w, 1/2 - w*omega, 0, 0, -w], [1/2 - 2*w*c, 1/2, 2*w*(1+conjugate(omega)), 0, -w, -w*conjugate(omega)], [(1/2)*(1 + omega) - w*(conjugate(omega) + omega^2), -w*(1 + omega), 2*w, 0, 0, 0], [1, 0, 0, 0, 0, 0], [(1/2)*omega + 2*w, 2*w*omega - w, 1/2 - w*omega, 0, 0, -w])];

- Introduce new variables, $cs$ and $ss$, for $\cos(\pi K / 16)$ and $\sin(\pi K / 16)$.
- $Cre := \text{diag}(1, 1, cs - 3*ss)$;
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.

\[ \begin{bmatrix} 1/4 & 1 & 0 \\ 0 & 1/4 & 1 \\ 0 & 0 & 1/4 \end{bmatrix} \]

**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

\[ \text{ButterCharpoly} := \text{subs}(cs = \sqrt{d}, \text{collect( subs( cos(m*Pi/K) = c, expand(charpoly(Butter00re, lambda),trig)), lambda))} \]

\[ \text{ButterCharpoly} := \frac{\lambda^3}{\lambda^3 + \frac{1}{192}d^\frac{3}{4} - \frac{37}{48}d^\frac{3}{2} + \frac{7}{64}d - \frac{1}{3}d^2} \]

\[ \text{Discr} := \text{simplify( (pc/3)^3 + (qc/2)^2)} \]

\[ \text{Discr} := \text{factor(subs( w = 1/16, Discr))} \]

\[ \text{plot(Discr, d = 0..1)} \]

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

\[ \text{DiscrFactor4} := \text{factor(Discr/(lcoeff(Discrs)*(d-1/4)*(d-1)^2*d))} \]

\[ \text{plot(DiscrFactor4, d=0..1)} \]
The case of three real roots

Find the interesting root

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

> evalf(DiscrRoot);

We observe that the discriminant has four roots: 0, 1, 1/4 and approx. 0.849985269; 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 DiscrRoot..1, negative on 1/4..DiscrRoot

We conclude that the second root is the largest.

> evalf(DiscrRoot); 0.8498612039

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8498612039; 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.

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8498612039; 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 DiscrRoot. 1, negative on 1/4. DiscrRoot

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

> N := sqrt(-factor(pe)/3):

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

> N := sqrt(-factor(pe)/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 discriminant is negative when d < -768/144 = -5.357142857, positive when d > 0, and zero at d = 0.

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

> N := sqrt(-factor(pe)/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 discriminant is negative when d < -768/144 = -5.357142857, positive when d > 0, and zero at d = 0.

We conclude that the second root is the largest.

> evalf(DiscrRoot); 0.8498612039

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8498612039; 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 DiscrRoot. 1, negative on 1/4. DiscrRoot

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

> N := sqrt(-factor(pe)/3):

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

> N := sqrt(-factor(pe)/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 discriminant is negative when d < -768/144 = -5.357142857, positive when d > 0, and zero at d = 0.

We conclude that the second root is the largest.

> evalf(DiscrRoot); 0.8498612039

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8498612039; 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 DiscrRoot. 1, negative on 1/4. DiscrRoot

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

> N := sqrt(-factor(pe)/3):

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

> N := sqrt(-factor(pe)/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 discriminant is negative when d < -768/144 = -5.357142857, positive when d > 0, and zero at d = 0.

We conclude that the second root is the largest.

> evalf(DiscrRoot); 0.8498612039

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8498612039; 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 DiscrRoot. 1, negative on 1/4. DiscrRoot

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

> N := sqrt(-factor(pe)/3):

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

> N := sqrt(-factor(pe)/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 discriminant is negative when d < -768/144 = -5.357142857, positive when d > 0, and zero at d = 0.

We conclude that the second root is the largest.

> evalf(DiscrRoot); 0.8498612039

We observe that the discriminant has four roots: 0, 1/4 and approx. 0.8498612039; 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 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 < \( x \). We see that the characteristic 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.

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

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

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

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

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

For plots, get the expressions for the roots

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

**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..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.

```maple
> R := sqrt(-pe/3): phi := arccosh( abs(qe)/(2*abs(R)^3));
```
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 \( y' + r(d)y^2 + s(d)y + t(d) = 0 \), and setting \( \frac{dy}{dd} \) to zero, we observe that at a point \( d_0 \) where the derivative is zero, the value \( y(d_0) \) satisfies the equation
\[
\frac{2}{dd}y^3 + \frac{2}{dd}y^2 + \frac{2}{dd}y = 0.
\]
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.

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 \( d \);

\[ dEquation := \text{finduni}(d, [\text{subs}(\lambda = y, \text{ButterCharpoly}), \text{diffCharpoly}]); \]

Now find a rational interval for \( d \);

\[ \text{IntervalDominantMax} := \text{op}(	ext{realroot}(dEquation, 1/10^7)); \]

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 used the algebraic value of the root returned by \text{solve}, rather than transcendental equation, because interval \text{apply} does not work properly for hyperbolic functions.

\[ \text{eval(IntervalDeriv(0.9))}; \]

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

\[ \text{Block 0} \]

\[ \text{Block 1} \]

\[ \text{Block 2} \]
Valences 4,5
K = 4; Check which formulas to use.
\[
\begin{pmatrix}
1 & 16 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1
\end{pmatrix}
\]

\[\text{The largest eigenvalue is in the first block.}\]
\[\text{inapply(DiscrRoot); inapply(cos(Pi/4)^2); inapply(cos(2*Pi/4)^2);} \]
\[\rightarrow () \begin{pmatrix}
0.84868120391246120009 \\
0.84868120391246120352
\end{pmatrix}
\]
\[\rightarrow () \begin{pmatrix}
0.658927841848274661 \\
0.6589278418482746486
\end{pmatrix}
\]
\[\text{K = 5; inapply(DiscrRoot); inapply(cos(Pi/5)^2); inapply(cos(2*Pi/5)^2);} \]
\[\rightarrow () \begin{pmatrix}
0.84868120391246120009 \\
0.84868120391246120352
\end{pmatrix}
\]
\[\rightarrow () \begin{pmatrix}
0.65450849718743711227 \\
0.65450849718743711227
\end{pmatrix}
\]
\[\rightarrow () \begin{pmatrix}
0.09549150281252628786 \\
0.09549150281252628786
\end{pmatrix}
\]
\[\text{We do not have to check the eigenvalue of the second block: for it, } d < 1/4, \text{ therefore, the magnitude of the largest eigenvalue is also less than 1/4.}\]

\[\text{K = 6; Check which formulas to use.}\]
\[\text{We have established that the magnitude of the largest eigenvalue decreases as the function of } d \text{ when } d > 0.6700423 = d_0; \text{ if } d = d_0 \begin{pmatrix}
\cos \frac{\pi}{K} \\
\frac{m}{2}
\end{pmatrix}
\]
\[\text{We see that in this case the eigenvalue of the second block is larger:}\]
Eigenvectors

Find the eigenvectors in two steps. Using the special structure of the matrix \[
\begin{pmatrix}
B_{00} & 0 \\
B_{10} & B_{11}
\end{pmatrix}
\] and the fact that we are interested in the eigenvectors which are also eigenvalues of the subblock \( B_{11} \), we find the eigenvector as 
\[
\psi \cdot (B_{11} - \lambda I)^{-1} B_{10} \psi,
\]
where \( \psi \) is the eigenvector of \( B_{11} \). We also use the fact that in all cases of interest, the eigenvalues of \( B_{00} \) are not eigenvalues of \( B_{11} \).

To compute an eigenvector of \( B_{11} \), check first that the second and third lines of the matrix are always independent; the second component of the cross product is not zero, because 
\[
\frac{1}{2} \leq \lambda.
\]

Compute the first part of the vector:
\[
\psi_0 := \text{sub}(\text{map}(\text{simplify}, \text{linsolve}(
\text{submatrix}(\text{eval}(\text{Butterfly00}-\text{lambda}^*() ) , [2...3, 1..3],[0,0] )));
\]
\[
\psi_0 := \begin{pmatrix}
16 \\
16 \\
-1-8 \lambda
\end{pmatrix}
\]

Butter11lambda := \text{eval}(\text{Butterfly11} - \text{lambda}^*() )

Compute the second part of the eigenvector:
\[
\psi_1 := \text{map}(\text{simplify}, \text{sub}([a^3 = a^{1-c^2}, a^2 = 1-c^2], \text{map}(\text{expand}, \text{eval}(-\text{inverse}(\text{Butter11lambda}) \cdot \text{Butterfly10} \cdot \psi_0 ))));
\]
\[
\psi_1 := \begin{pmatrix}
16 \\
16 \\
-1-8 \lambda
\end{pmatrix}
\]

Put the two parts of the vector together and simplify notation
\[
\psi := \text{map}(\text{simplify}, \text{seq}(\psi_0[1], i=1...3), \text{seq}(\psi_1[i], i=1...3)));
\]

Check if the expression makes sense for \( K = 3 \): 
\[
\psi := \text{map}(\text{expand}, \text{sub}([\text{lambda} = 1/2, c = 1/2, s = \sqrt{3}/2], \text{eval}(\text{ButterEigenvect})));
\]
\[
\psi := \begin{pmatrix}
1/2 \\
1/4 \\
1/2
\end{pmatrix}
\]

Code generation

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

Float Eigenvalue\( (\text{int} \ K) \) computes the eigenvalues, void Eigenvector\( \text{real}(\text{Float} \ c, \text{Float} \ \lambda, \text{Float}^* \ \text{EvRe}) \) initializes an array for the real part of the complex eigenvector, void Eigenvector\( \text{imag}(\text{Float} \ c, \text{Float} \ \lambda, \text{Float}^* \ \text{EvIm}) \) initializes the array for the complex part of the eigenvector. 

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\( \text{fopen} \) and \( \text{fdiv} \) (see \( \text{ConvertToFloat} \) for details).

C \( \text{OutputFile} := \text{`butterfly.cpp'}; \)
C \( \text{MakeClassHeader} (\text{OutputFile}, \text{`Butterfly'}, 2, 4, 5, \text{RegButterfly}); \)

Code generation for eigenvalues

To show that the scheme produces \( C_1 \) surfaces for valences 4,5,7, and not \( C_1 \) 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.

ComputeEigenvvalues\( (\text{N}, \text{eps}, \text{fname}) \) 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 interv al \([1/4, 1]\) in the range of interest. This function computes eigenvalues up to valence\( \text{N} \) verifying that the precision is no less than\( \text{eps}. \)

C \( \text{ComputeEigenvvalues} (\text{N}, \text{eps}, \text{fname}) \) { \text{compute eigenvalues numerically with verified; } \text{write a function with a large table into a file.} \}
C \( \text{Check if the expression makes sense for } K = 3; \)
C \( \text{Check if the expression makes sense for } K = 6; \)
C \( \text{Check if the expression makes sense for } K = 6; \)
C \( \text{Check if the expression makes sense for } K = 6; \)
C \( \text{Check if the expression makes sense for } K = 6; \)
for K from 4 to N do
if (K - 4) mod 100 = 0 then print(K);
fi;
Digits := 25;
approxEV := fsolve(expandedCharpoly, lambda, lambda=0.25..1);
# check that the precision is at least eps
if op(2, intervCharpoly(approxEV-eps, dK)) > 0
or op(1, intervCharpoly(approxEV+eps, dK)) < 0 then
ERROR('fsolve precision failure for K = ', K);
fi;
Digits := 15; # need to get the right number of digits for printing
fprintf(fname, 'CONST64(%d),CONST64(%d),CONST64(%d),
', op(1,evalf(approxEV-eps)),
op(1,evalf(approxEV+eps)),10^(-op(2,evalf(approxEV))));
Digits := 25;
end;

fprintf(fname, `CONST64(0)};
return Float(EV[3*K],EV[3*K+1])/Float(EV[3*K+2]) ;
}`

Generate eigenvalues

ComputeEigenvalues( 4500, 1e-10, OutputFile):

optEV1 := 'optimize/makeproc'([map(ConvertToFloat,
[optimize( AbsDominantEV1)])],parameters=[d]):

Fix a problem with C code generation: no return value is generated, if the last statement is assignment

optEV1body := procbody(optEV1):
EVstatseq := op(5,optEV1body) := 'statseq'(op(EVstatseq), op(1,op(-1,EVstatseq))):
optEV1 := procmake(subsop(5=EVstatseq, optEV1body)):
S := []: C(optEV1,ansi):
for i from 1 to vectdim(S) do
fprintf(OutputFile, replaceall(op(i,S),`double`,`Float`)):
od:

Second version for valences >= 8;

optEV2 := 'optimize/makeproc'([map(ConvertToFloat,
[optimize( op(1,
[solve(ButterCharpoly,lambda)]))]),parameters=[d]):

optEV2body := procbody(optEV2):
EVstatseq := op(5,optEV2body) := 'statseq'(op(EVstatseq), op(1,op(-1,EVstatseq))):
S := []: C(optEV2body,ansi):
for i from 1 to vectdim(S) do
fprintf(OutputFile, replaceall(op(i,S),`double`,`Float`)):
od:

This function is used to construct interval eigenvectors "at infinity"; assume d sufficiently close to 1, so that ( )

λ
c

decreases.

virtual void EigenvalueRange( Float c, Float& lambdamin, Float& lambdamax) {
Float d = FR(1,2)*(c + Float(1)); assert( (exactfloor(d)).as_double() > 0.676);
if( exactceil(c)  < exactfloor( sqrt(Float(2))/Float(2)) )
lambdamax = optEV1(d);
else lambdamax = optEV2(d); lambdamin = FR(1,4);
}


This is simple enough for the Maple function.

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^t[1conjugate(omega)],[5,0,-w,-w,conjugate(omega)],
[1/2 + 2*w^omega, 2*w = w^omega, 1/2 - 2*w^conjugate(omega),0,-w,0],
[(1/2)*omega + 2*w^2*omega*w, 1/2 = w^omega,0,0,0,0]);

This is simple enough for the Maple function.
jordan(ModButter, 'P');

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 :=

Code generation