TECHNISCHE
UNIVERSITÄT
DARMSTADT

A New Approach to the Construction of Subdivision Algorithms: The MATLAB Software Package

Manual

by
Alexander Dietz

TUdatalib 2025

Contents

This software package is the implementation of the present dissertation [Die25b]:

> Alexander Dietz. *Ein neuer Ansatz zur Konstruktion von Subdivisionsalgorithmen*
> Dissertation. Darmstadt: TU Darmstadt. 2025 DOI: https://doi.org/10.26083/tuprints-00030194

and the corresponding English translation [Die25a]:

> Alexander Dietz. *A New Approach to the Construction of Subdivision Algorithms*
> TUprints. Darmstadt. 2025 DOI: https://doi.org/10.26083/tuprints-00030195

1 Quick Use

For quick use, there is the function

```
[S,Lattice] = computeSubdivisionMatrix(Input,Degree,MatrixSize,varargin)
```

This function automatically selects the appropriate algorithm for creating the subdivision algorithms. The following must be considered:

- Input: Must either be a natural number greater than 2 (for the two-dimensional case) or one of the inputs listed below, such as adjacency matrix, edge matrix, or face matrix (for the three-dimensional case).
- Degree: Must be 2 (or 'quadratic' or 'Quadratic') for the quadratic case or 3 (or 'cubic' or 'Cubic') for the cubic case.
- MatrixSize: Must be 0 (or 'initial' or 'Initial') for initial elements or 1 (or 'big' or 'Big') for larger structures.
- If Variant 1 is to be used in the quadratic case, the command must include the optional input

  ```
  computeSubdivisionMatrix(Input,Degree,MatrixSize,'mu',value)
  ```

  with $0 \leq$ `value` $< 1/2$ (even if mu is not supportet in the 2D case).
- Also, all optional inputs described below can be used.

Examples for the command are:

```
S=computeSubdivisionMatrix(5,2,0,'mu',1/4);                %2D | quadratic | initial element | V1
S=computeSubdivisionMatrix(6,'quadratic',0);               %2D | quadratic | initial element | V2
S=computeSubdivisionMatrix(3,2,'Big','mu',1/4,'Status');   %2D | quadratic | bigger structure | V1
S=computeSubdivisionMatrix(4,2,1,'PreventInputCheck');     %2D | quadratic | bigger structure | V2
S=computeSubdivisionMatrix(7,3,'initial');                 %2D | cubic | initial element
S=computeSubdivisionMatrix(4,'cubic','big');               %2D | cubic | bigger structure
```

and with

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 7 & 8 & 6 \\ 2 & 3 & 7 & 0 & 0 \\ 3 & 4 & 8 & 7 & 0 \\ 4 & 5 & 6 & 8 & 0 \\ 5 & 1 & 6 & 0 & 0 \end{bmatrix}$$

for the 3D case:

```
S=computeSubdivisionMatrix(F,2,0,'mu',1/4);                  %3D | quadratic | initial element | V1
S=computeSubdivisionMatrix(F,"Quadratic",0);                 %3D | quadratic | initial element | V2
S=computeSubdivisionMatrix(F,2,1,'mu',1/4);                  %3D | quadratic | bigger structure | V1
S=computeSubdivisionMatrix(F,2,"big",'MaxIterations',50);    %3D | quadratic | bigger structure | V2
S=computeSubdivisionMatrix(F,3,'Initial');                   %3D | cubic | initial element
S=computeSubdivisionMatrix(F,'cubic','big');                 %3D | cubic | bigger structure
```

2 Disclaimer

This manual is intended to provide an overview of the included code and to make it as easy as possible to use. It does not explain the mathematical background or the detailed functionality of the code. For that, please refer to the associated dissertation, which also includes many examples and illustrations.

Additionally, please note that this manual has been linguistically edited using ChatGPT.

3 Contact & License

For questions, comments, or to report bugs, you can contact the author anytime via the following email:

alexander_dietz@t-online.de

The Matlab code may be used under the license:

CC-BY 4.0 International

as long as the following dissertation:

Alexander Dietz. *Ein neuer Ansatz zur Konstruktion von Subdivisionsalgorithmen*
Dissertation. TU Darmstadt. 2025 DOI: https://doi.org/10.26083/tuprints-00030194

and the corresponding English translation:

Alexander Dietz. *A New Approach to the Construction of Subdivision Algorithms*
TUprints. Darmstadt. 2025 DOI: https://doi.org/10.26083/tuprints-00030195

is cited.

4 Demo Files

There is one demo file for each of the categories: generalized quadratic and generalized cubic B-spline subdivision. These files are designed to offer an easy introduction to how the software works and what it can do. It is recommended to run both demo files

`DemoQuadratic` and `DemoCubic`

once before you start working with the software. There are also two live demo files

`DemoQuadraticLive` and `DemoCubicLive`

which have the same content. It is recommended to have a look at both live demo files.

5 Goal of the Algorithms

The goal of this software package is to create subdivision matrices for generalized quadratic and cubic B-spline subdivision. They produce subdivision matrices for subdivision surfaces as well as subdivision volumes. The subdivision matrices define refinement rules for arbitrary combinatorial structures. Irregular points and edges are supported in all cases.

All generated subdivision matrices have a valid eigenstructure: they have a subdominant eigenvalue of $1/2$ with multiplicity two (for subdivision surfaces) and multiplicity three (for subdivision volumes).[1] Moreover the central part of the structure forms a convex polytope (or in the cubic case, a set of cubes whose outer faces form a central polytope). More details are available in the corresponding dissertation.

6 Input and Output Parameters

Many of the functions use the same input and output parameters. To avoid repeating the same explanations, the common parameters are described here:

`n`

The parameter `n` is the valency of the extraordinary vertex and used for two-dimensional subdivision. It must satisfy

$$\texttt{n} \in \{3, 4, 5, \ldots\}.$$

In the quadratic case, `n` gives the number of corners of the (central) polytope. In the cubic case, it gives the number of quadrilateral faces that meet at the (central) point.

`Input`

The parameter `Input` is used for volumetric subdivision. In the quadratic case, it describes the structure of the central polytope. In the cubic case, it describes the dual structure of the cubes around the central point.

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a 3-connected planar graph representing the central polytope or the dual structure of the cubes around the central point. The parameter `Input` can take one of the following three formats:

- The adjacency matrix
  $$A \in \{0, 1\}^{|\mathbf{V}| \times |\mathbf{V}|}$$
  of **G**.
- A list of edges
  $$E \in \{1, \ldots, |\mathbf{V}|\}^{|\mathbf{E}| \times 2}.$$
  Each row in E represents one edge in **G**. The two entries in each row are the nodes connected by that edge.
- A list of facets
  $$F \in \{0, 1, \ldots, |\mathbf{V}|\}^{(2-|\mathbf{V}|+|\mathbf{E}|) \times m},$$
  where m is the largest number of nodes in any one facet of **G**. Each row in F represents one facet of **G**. Each entry in a row is a node. Two nodes are connected by an edge if they are next to each other in the row. The last node and the first node are also connected. Because the number of nodes per facet can vary, the number of columns in F is set to the size of the largest facet. The first entry of each row contains a node. Extra entries are filled with zeros.

`mu`

The parameter `mu` is only used in Variant 1 of the quadratic case. It defines the desired subsubdominant eigenvalue. Valid values are $\texttt{mu} \in [0, 1/2)$, and it is recommended to choose $\texttt{mu} \in [0, 1/4]$.

[1] This has been proven for the quadratic case and empirically verified for the cubic case. If you discover a combinatorial structure where this is not true in the cubic case, please report it using the email address above.

`varargin`

There are several additional optional inputs for the subdivision algorithms. These are explained in Section 8.

`S`

The parameter `S` is the output of the functions that create subdivision matrices. `S` is always a real square matrix and represents the subdivision matrix that was generated based on the given input.

`Controlpoints`

The parameter `Controlpoints` refers to a matrix of control points. Each row in `Controlpoints` is one control point in $\mathbb{R}^d$ with $d \in \mathbb{N}$. The matrix contains m control points in total. So we have

$$\texttt{Controlpoints} \in \mathbb{R}^{m \times d}.$$

`Lattice`

The parameter `Lattice` describes the structure of three-dimensional grids. Suppose a grid consists of m control points and k polytopes. Then

$$\texttt{Lattice} \in \{0,1\}^{m^2 \times k}.$$

Each column in `Lattice` is an adjacency matrix. Its dimensions cover all control points in the grid. Each individual adjacency matrix can be reconstructed using

$$\texttt{reshape(Lattice(:,i),[m,m])}.$$

In the quadratic case, each adjacency matrix in `Lattice` must represent a 3-polytope. In the cubic case, each one must represent a cube. It is recommended to use sparse matrices.

7 Functions for Generating Subdivision Matrices

The software package includes a total of 12 different functions for generating subdivision matrices. These functions differ as follows:

- Functions containing `quadratic` generate subdivision matrices for the quadratic case, while those containing `cubic` generate matrices for the cubic case.
- Functions containing `bi` generate subdivision matrices for surface subdivision, while those containing `tri` are used for volumetric subdivision.
- Functions without the ending `Big` generate subdivision matrices for central structures. In the quadratic case, these map one polytope to another. In the cubic case, they map a structure of quadrilaterals (for surfaces) or cubes (for volumes) sharing a central point to a new structure of quadrilaterals or cubes.
  Functions with the ending `Big` generate subdivision matrices for larger structures. In the quadratic case, the matrix maps a ring (or a shell in the volumetric case) to a new ring (or shell). The size is chosen so that one full ring of regular B-spline patches (or volumes) can be evaluated. In the cubic case, the matrix maps a double ring (or double shell in the volumetric case) to another double ring (or shell). The size is chosen so that two full rings of regular B-spline patches (or volumes) can be evaluated. More background on the double ring or shell structure can be found in the associated dissertation.
- For the quadratic case, two variants have been developed. Functions containing `V1` use Variant 1 to compute subdivision matrices, while those with `V2` use Variant 2. In Variant 1, the subsubdominant eigenvalue can be freely selected using the parameter `mu`. Variant 2 guarantees no negative entries and provides a more natural construction in semi-irregular regions (such as irregular edges).

For the surface case, the following 6 functions are available:

```
                  [S]=computeBiQuadraticSubdivisionMatrixV1(n)
     [S,CR,PrimalPoints]=computeBiQuadraticSubdivisionMatrixV2(n)
                  [S]=computeBiQuadraticSubdivisionMatrixV1Big(n)
                  [S]=computeBiQuadraticSubdivisionMatrixV2Big(n)
                  [S]=computeBiCubicSubdivisionMatrix(n)
                  [S]=computeBiCubicSubdivisionMatrixBig(n)
```

The properties of these functions are described below:

- The input for all these functions is the number of corners of the (central) polytope in the quadratic case, or the number of quadrilaterals around a (central) point in the cubic case. All functions return the corresponding subdivision matrix.
- The functions `computeBiQuadraticSubdivisionMatrixV1(n)` and `computeBiQuadraticSubdivisionMatrixV2(n)` return cyclic matrices.
- In `computeBiCubicSubdivisionMatrix(n)`, the row and column order is as follows: the first n rows/columns represent the corner points, the next n rows/columns represent the edge points, and the final row/column represents the central point.
- The functions `computeBiQuadraticSubdivisionMatrixV1Big(n)` and `computeBiQuadraticSubdivisionMatrixV2Big(n)` also return cyclic matrices. These are made up of blocks of 9 points, with each block corresponding to one point of the central polytope.
- In `computeBiCubicSubdivisionMatrixBig(n)`, the first row/column represents the central point. The remaining rows/columns are cyclic and organized in 30×30 blocks. Each block consists of 5×6 points. The layout follows the 3×4 pattern from [PR08].

For the volumetric case, the following 6 functions are available:

```
          [S]=computeTriQuadraticSubdivisionMatrixV1(Input,mu,varargin)
          [S]=computeTriQuadraticSubdivisionMatrixV2(Input,varargin)
  [S,Lattice]=computeTriQuadraticSubdivisionMatrixV1Big(Input,mu,varargin)
  [S,Lattice]=computeTriQuadraticSubdivisionMatrixV2Big(Input,varargin)
  [S,Lattice]=computeTriCubicSubdivisionMatrix(Input,varargin)
  [S,Lattice]=computeTriCubicSubdivisionMatrixBig(Input,varargin)
```

The properties of these functions are described below:

- The `Input` for all six functions is as described in Section 6.
- For Variant 1 of `computeTriQuadraticSubdivisionMatrixV1` and `computeTriQuadraticSubdivisionMatrixV1Big`, the parameter `mu` defines the desired subsubdominant eigenvalue.
- All six functions can also accept optional parameters via `varargin`. A summary of these options can be found in Section 8.
- The order of entries in the subdivision matrices `S` from `computeTriQuadraticSubdivisionMatrixV1` and `computeTriQuadraticSubdivisionMatrixV2` follows the input. The order of the adjacency, edge, or face matrix is preserved.
- The order of entries in the subdivision matrices `S` from the other four functions is encoded in the output parameter `Lattice`. This contains the adjacency matrices of all polytopes in the structure.

8 Optional Input Parameters `varargin`

There are several optional input parameters available for the six volumetric functions. These are explained below:

`'Status'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix(Input,'Status')
```

This option enables status messages during the construction of the subdivision matrix. These messages describe the steps of the process.

`'Visualization'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix(Input,'Visualization')
```

This option generates plots during the construction of the subdivision matrix. These plots illustrate both the process and the result. More background is available in the associated dissertation.

`'PreventInputCheck'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix(Input,'PreventInputCheck')
```

With this option, the algorithm skips the check whether the input structure consists of valid polytopes. This can speed up computation, but if the input is invalid, helpful error messages might be missing.

`'Tolerance'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix(Input,'Tolerance',1e-7)
```

This option allows you to change the precision of the numerical processes used in the algorithm. The default value is 10^{-13}. This means the processes will run until a precision of 10^{-13} is reached. Note that this only applies to local steps and does not guarantee the overall output is accurate to 10^{-13}.

`'MaxIterations'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix(Input,'MaxIterations',50)
```

This option lets you set the maximum number of steps in the numerical processes. The default is 100, meaning each local process will try up to 100 iterations before stopping.

`'DrawAfterXKites'`

An example command using this option is:

```
[S,Lattice]
=computeTriCubicSubdivisionMatrix(Input,'Visualization','DrawAfterXKites',4)
```

When visualizing the graph, kites are drawn step by step. One kite is shown per iteration by default. With this option, you can specify how many kites should be drawn at once. This is useful because for large polytopes, drawing each kite separately can take a long time. The default value is 1.

`'PrintImages'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix
(Input,'Visualization','PrintImages','Test','.png')
```

This option exports the plots created by the `'Visualization'` command. The first additional argument `'Test'` sets the file name prefix, and the second `'.png'` defines the suffix and with this also the file format.

`'View'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrix(Input,'Visualization','View',[34,65])
```

This option rotates the generated plot to the angles given by `[34,65]`. This is equivalent to the MATLAB command:

```
view([34,65])
```

`'KeepFaceOrder'`

An example command using this option is:

```
[S,Lattice]=computeTriCubicSubdivisionMatrixBig(FaceMatrix,'KeepFaceOrder')
```

This option is only available for the `Big` variants and only when the input `FaceMatrix` is a face matrix. It ensures that the order of faces from `FaceMatrix` is preserved in the row and column order of the matrix `S`.

9 Quality-of-Life Functions

The software package includes five additional core functions that are important for applying the refinement. These functions are explained below.

`RefineTriCubicLatticeUniform`

The command is:

```
[ControlpointsNew,LatticeNew] = RefineTriCubicLatticeUniform(Controlpoints,Lattice)
```

This function takes a given `Lattice` with associated `Controlpoints` and refines it uniformly and globally into a finer `LatticeNew` with new `ControlpointsNew`. All adjacency matrices (columns of the variable) in `Lattice` must represent cubes. The adjacency matrices in `LatticeNew` also all represent cubes. The refinement is performed using the function `computeTriCubicSubdivisionMatrix` for each initial element. Background information is provided in the associated dissertation. An example can be found in the corresponding demo file.

RefineTriQuadraticLatticeUniformV1 and RefineTriQuadraticLatticeUniformV2

The commands are:

```
 [ControlpointsNew,LatticeNew,LatticeVolume,LatticeFaces,
 LatticeEdges,LatticeVertices,IsInnerVertex,ValenceInnerVertex]
=RefineTriQuadraticLatticeUniformV1(Controlpoints,Lattice,mu)
```

and

```
 [ControlpointsNew,LatticeNew,LatticeVolume,LatticeFaces,
 LatticeEdges,LatticeVertices,IsInnerVertex,ValenceInnerVertex]
=RefineTriQuadraticLatticeUniformV2(Controlpoints,Lattice)
```

These two functions refine a given `Lattice` with associated `Controlpoints` uniformly and globally into a finer `LatticeNew` with new `ControlpointsNew`. All adjacency matrices (columns of the variable) in `Lattice` must represent 3-polytopes. The adjacency matrices in `LatticeNew` will also all represent 3-polytopes. The refinement is carried out using the function `computeTriQuadraticSubdivisionMatrixV1` or `computeTriQuadraticSubdivisionMatrixV2` for each initial element. In Variant 1, the subsubdominant eigenvalue can also be set using the parameter `mu`. More background is given in the associated dissertation. An example is provided in the corresponding demo file.

In addition, these functions provide several optional output parameters. The parameters `LatticeVolume`, `LatticeFaces`, `LatticeEdges`, and `LatticeVertices` specify which parts of `LatticeNew` came from volumes, facets, edges, and vertices of the original `Lattice`. The parameter `IsInnerVertex` indicates which of the `ControlpointsNew` are inner vertices, and `ValenceInnerVertex` gives the valence of these inner vertices. An example is included in the corresponding demo file.

plotTriCubicBSplineLattice and plotTriQuadraticBSplineLattice

The commands are:

```
[] = plotTriCubicBSplineLattice(Controlpoints,Lattice,varargin)
```

and

```
[] = plotTriQuadraticBSplineLattice(Controlpoints,Lattice,varargin)
```

These two functions plot the evaluable part of a `Lattice` along with its corresponding `Controlpoints`. The function first checks which part of the structure can be evaluated as a trivariate tensor-product B-spline. This B-spline cube[2] is then plotted by evaluating each of its facets as a two-dimensional B-spline.

Both functions support two optional input parameters, described below:

- `'PointsPerCube'` An example command using this parameter is:

  ```
  [] = plotTriCubicBSplineLattice(Controlpoints,Lattice,'PointsPerCube',2)
  ```

  This parameter controls how many points per dimension are used to evaluate the trivariate B-spline cubes. For example, if set to 2, each facet will be plotted using 2×2 points. If set to 3, 3×3 points are used, and so on. The default value is 7.

[2]Here, “cube” does not mean a convex geometric cube. It refers to a domain of the form $[0, 1]^3$ being used to evaluate a trivariate B-spline.

- `'ColorTable'` With the matrix

$$C = \begin{bmatrix} 0.8 & 1 & 0.8 \\ 0.7 & 0.9 & 0.7 \\ 0.6 & 0.8 & 0.6 \\ 0.5 & 0.7 & 0.5 \\ 0.4 & 0.6 & 0.4 \\ 0.3 & 0.5 & 0.3 \\ 0.2 & 0.4 & 0.2 \\ 0.1 & 0.3 & 0.1 \\ 0.0 & 0.2 & 0.0 \\ 0 & 0.5 & 0 \\ 0.5 & 1 & 0.5 \\ 0.4 & 0.9 & 0.4 \\ 0.3 & 0.8 & 0.3 \\ 0.2 & 0.7 & 0.2 \\ 0.1 & 0.6 & 0.1 \end{bmatrix}$$

  you can use the following command as an example:

  ```
  [] = plotTriCubicBSplineLattice(Controlpoints,Lattice,'ColorTable',C)
  ```

  To help visually separate different B-spline cubes, different colors are used. By default, 15 shades of green are generated. After plotting 15 cubes, the 16th one reuses the first color, and so on. You can customize the colors with a matrix $C \in [0,1]^{n\times 3}$. Each row of C defines an RGB color with values in $[0,1]$. You can supply as many color rows as you like.

10 Other Useful Functions in the Software Package

The software package includes several other useful functions, which are described below:

`computeFaceMatrix`

The command is:

```
[FaceMatrix] = computeFaceMatrix(AdjacencyMatrix,Inputcheck)
```

This function generates a face matrix `FaceMatrix` from an adjacency matrix `AdjacencyMatrix`. The parameter `Inputcheck` is a boolean that specifies whether to check if the adjacency matrix represents a planar and 3-connected graph (`true`) or not (`false`). For example, with

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

you can use the following command:

```
[FaceMatrix] = computeFaceMatrix(A,true)
```

`computePrismFaceMatrix`

The command is:

```
[FaceMatrix] = computePrismFaceMatrix(n)
```

This function generates the face matrix of an `n`-sided prism. The parameter `n` must be in the set $\{3, 4, 5, \ldots\}$. Example:

```
[FaceMatrix] = computePrismFaceMatrix(5)
```

computeTrapezohedronAdjacencyMatrix

The command is:

```
[A] = computeTrapezohedronAdjacencyMatrix(n)
```

This function generates the adjacency matrix of an `n`-gonal trapezohedron. The input `n` must be in the set $\{3, 4, 5, \ldots\}$. Example:

```
[A] = computeTrapezohedronAdjacencyMatrix(3)
```

Construct3Polytope

The command is:

```
[PointCoordinates3D,KiteAll,StatusString,statusCounter]
= Construct3Polytope(Input,varargin)
```

This function generates a convex polytope from the input as described in Sections 6 and 8. All edges of the polytope lie tangentially on the unit sphere, and the center of all contact points lies at the origin. The polytope also preserves all symmetries of the input graph structure. A corresponding dual polytope with the same properties is also constructed. See Theorem 3.25 of the associated dissertation for more details.

The output `PointCoordinates3D` is an $m \times 3$ matrix. Each row is a point in $\mathbb{R}^3$. The matrix contains three categories of points: the first category are the vertices of the polytope, the second the tangent points, and the third the vertices of the dual polytope.

The output `KiteAll` is a $k \times 4$ matrix. It shows which points form the facets in the 2D drawing of the quad graph (combining primal, dual, and tangent points). Each facet is a kite. More information is available in the dissertation.

The outputs `StatusString` and `statusCounter` return the status messages and their number when the optional `'Status'` input is used. These are used internally by other functions and are not relevant for general users.

Using the above matrix A, an example call is:

```
[PointCoordinates3D,KiteAll,StatusString,statusCounter]= Construct3Polytope(A)
```

CreateTestScenario

The command is:

```
[DualAdjacency,PolygonFound] = CreateTestScenario(n,maxVertices)
```

This command creates a random adjacency matrix with up to `n` facets and up to `maxVertices` vertices. If successful, the result is returned in `DualAdjacency`. Whether the generation was successful is indicated by the boolean `PolygonFound`. An example for this command is:

```
[DualAdjacency,PolygonFound] = CreateTestScenario(30,50)
```

FaceToAdjacencyMatrix

The command is:

```
[AdjacencyMatrix] = FaceToAdjacencyMatrix(FaceMatrix)
```

This command converts a face matrix `FaceMatrix` into an adjacency matrix `AdjacencyMatrix`. For example, with

$$F = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}$$

the following call can be used:

```
[AdjacencyMatrix] = FaceToAdjacencyMatrix(F)
```

is3Connected

The command is:

```
[Result] = is3Connected(AdjacencyMatrix)
```

This command checks whether the graph defined by the adjacency matrix `AdjacencyMatrix` is 3-connected. It returns `true` if it is, and `false` otherwise. An Example with the above matrix A is:

```
[Result] = is3Connected(A)
```

PlanarityTest

The command is:

```
[Result] = PlanarityTest(AdjacencyMatrix)
```

This command checks whether the graph defined by the adjacency matrix `AdjacencyMatrix` is planar. It returns `true` if it is, and `false` otherwise. An Example with the above matrix A is:

```
[Result] = PlanarityTest(A)
```

References

[Die25a] Alexander Dietz. *A New Approach to the Construction of Subdivision Algorithms. English Translation of the Dissertation: Ein neuer Ansatz zur Konstruktion von Subdivisionsalgorithmen*. TUprints, 2025. DOI: `https://doi.org/10.26083/tuprints-00030195`.

[Die25b] Alexander Dietz. "Ein neuer Ansatz zur Konstruktion von Subdivisionsalgorithmen". PhD thesis. Darmstadt: TU Darmstadt, 2025. DOI: `https://doi.org/10.26083/tuprints-00030194`.

[PR08] Jörg Peters and Ulrich Reif. *Subdivision Surfaces*. 3 vols. Geometry and Computing. Springer Berlin, Heidelberg, 2008. DOI: `https://doi.org/10.1007/978-3-540-76406-9`.