Title: Tutorial: Monte Carlo methods in Dynamical Triangulations

Date: Jun 21, 2017 02:00 PM

URL: http://pirsa.org/17060079

Abstract:

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

WOLFRAM MATHEMATICA | STUDENT EDITION   Demonstrations | MathWorld | Wolfram Community | Help

Config

```
In[1]:= (* Provide the location of the randgeom executable *)
        (* On windows change randgeom to randgeom.exe *)
        programlocation = FileNameJoin[{NotebookDirectory[], "linux", "randgeom"}]

Out[1]= /home/timothy/Documents/web/homepage/randgeom/linux/randgeom

In[2]:= (* Test the program. If it return False, check the programlocation provided. If it still does not work, first test randgeom from the console. *)
        FileExistsQ[programlocation] && ListQ[RunThrough["'" <> programlocation <> "'", ""]]

Out[2]= True
```

Useful functions

```
In[3]:= (* the following just runs randgeom with the specified parameters and parses the output as Mathematica code *)
        generateMaps[type_, size_, number_] := RunThrough["'" <> programlocation <> "' -t" <> type <> " -s" <> ToString[size] <> " -n" <> ToString[number], ""];
        generateMap[type_, size_] := First@generateMaps[type, size, 1];

In[5]:= (* given a permutation p of {1,2,...,n}, cycles[p] gives the partition of {1,2,...,n} into cycles *)
        cycles[p_] := PermutationCycles[p, Identity];
        (* given a list plist of permutations, orbits[plist] gives the partition of {1,2,...,n} into orbits under the permutations *)
        orbits[plist_] := GroupOrbits@PermutationGroup[PermutationCycles /@ plist]

In[7]:= (* edges, vertices and faces correspond to cycles of halfedge-permutations *)
        edgecycles[map_] := cycles[map[[All, 3]]];
        facecycles[map_] := cycles[map[[All, 1]]];
        vertexcycles[map_] := cycles[map[[map[[All, 3]], 1]]];
        (* We may assign id's to the vertices of map according to their position in vertexcycles[map] *)
        halfedgeToVertexId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, vertexcycles[map], {2}]];
        halfedgeToFaceId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, facecycles[map], {2}]];

In[12]:= (* functions to construct a Mathematica Graph object *)
        uniqueEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToVertexId[map])];
        uniqueDualEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToFaceId[map])];
        mapGraph[map_] := With[{edges = uniqueEdges[map]}, Graph[Union @@ edges, #[[1]] ↔ #[[2]] & /@ edges, GraphLayout → None]];
        mapDualGraph[map_] := With[{edges = uniqueDualEdges[map]}, Graph[Union @@ edges, #[[1]] ↔ #[[2]] & /@ edges, GraphLayout → None]];
```

Plotting

```
In[16]:= (* the following is a bit of a hack to extract coordinates from GraphPlot3D's embedding of a graph *)
```

100%

Menu [Home] mcdt-part1.pdf example-analysi... [Inbox - timothy... 20:05

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

WOLFRAM MATHEMATICA | STUDENT EDITION — Demonstrations | MathWorld | Wolfram Community | Help

```
In[7]:= (* edges, vertices and faces correspond to cycles of halfedge-permutations *)
edgecycles[map_] := cycles[map[[All, 3]]];
facecycles[map_] := cycles[map[[All, 1]]];
vertexcycles[map_] := cycles[map[[map[[All, 3]], 1]]];
(* We may assign id's to the vertices of map according to their position in vertexcycles[map] *)
halfedgeToVertexId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, vertexcycles[map], {2}]];
halfedgeToFaceId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, facecycles[map], {2}]];

In[12]:= (* functions to construct a Mathematica Graph object *)
uniqueEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToVertexId[map])];
uniqueDualEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToFaceId[map])];
mapGraph[map_] := With[{edges = uniqueEdges[map]}, Graph[Union @@ edges, #[[1]] ↔ #[[2]] & /@ edges, GraphLayout → None]];
mapDualGraph[map_] := With[{edges = uniqueDualEdges[map]}, Graph[Union @@ edges, #[[1]] ↔ #[[2]] & /@ edges, GraphLayout → None]];
```

Plotting

```
In[16]:= (* the following is a bit of a hack to extract coordinates from GraphPlot3D's embedding of a graph *)
get3DGraphEmbedding[edg_, method_] := (VertexCoordinateRules /. Cases[GraphPlot3D[#[[1]] → #[[2]] & /@ edg, Method → method], _Rule, Infinity])[[Ordering@DeleteDuplicates[Join @@ (List @@@ edg)]]];

In[17]:= (* one can play with the options here to adapt the embedding *)
coordinates[map_] := get3DGraphEmbedding[uniqueEdges[map], {"SpringElectricalEmbedding", "InferentialDistance" → Automatic, "RepulsiveForcePower" → -2.4}];

In[18]:= (* assign colors to faces according to closeness centrality *)
colorFunction[x_] := ColorData["Rainbow"][Round[Max[Min[0.55 - 0.15 x, 1], 0], 1/40]];
centralityColors[map_] := colorFunction /@ Standardize@ClosenessCentrality[mapDualGraph[map]];

In[20]:= plotMap3D[map_, coor_] :=
 Graphics3D[{EdgeForm[None], {FaceForm[#[[1]]], Polygon[coor[[#[[2, ;; 3]]]]], If[Length[#[[2]]] == 4, Polygon[coor[[#[[2, {1, 3, 4}]]]]], {}]} & /@
   Transpose[{centralityColors[map], facecycles[map] /. halfedgeToVertexId[map]}]}, Boxed → False]

In[21]:= map = generateMap["C", 500];
plotMap3D[map, coordinates[map]]

(* Task: plot geometries of various sizes and models. What are the qualitative differences between the models? *)

(* Task: produce some nice pictures. To save a nice picture, one may use something like Export["picture.png",ImageCrop@Rasterize[plot,ImageSize→800,Background→None]] *)
```

Geodesic two-point function

```
(* Mathematica has built in support for graph distances: this returns a list of distances from all vertices to a randomly chosen vertex *)
distanceListFromRandomPoint[graph_] := GraphDistance[graph, RandomChoice@VertexList[graph]];
```

Menu [Home] mcdt-part1.pdf example-analysi... [Inbox - timothy... 20:05

100%

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

WOLFRAM MATHEMATICA | STUDENT EDITION

Demonstrations | MathWorld | Wolfram Community | Help

```
halfedgeToVertexId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, vertexcycles[map], {2}]];
halfedgeToFaceId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, facecycles[map], {2}]];
```

In[12]:=
```
(* functions to construct a Mathematica Graph object *)
uniqueEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToVertexId[map])];
uniqueDualEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToFaceId[map])];
mapGraph[map_] := With[{edges = uniqueEdges[map]}, Graph[Union @@ edges, #[[1]] ↔ #[[2]] & /@ edges, GraphLayout → None]];
mapDualGraph[map_] := With[{edges = uniqueDualEdges[map]}, Graph[Union @@ edges, #[[1]] ↔ #[[2]] & /@ edges, GraphLayout → None]];
```

Plotting

In[16]:=
```
(* the following is a bit of a hack to extract coordinates from GraphPlot3D's embedding of a graph *)
get3DGraphEmbedding[edg_, method_] := (VertexCoordinateRules /. Cases[GraphPlot3D[#[[1]] → #[[2]] & /@ edg, Method → method], _Rule, Infinity])[[Ordering@DeleteDuplicates[Join @@ (List @@@ edg)]]];
```

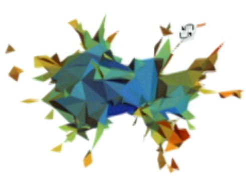
In[17]:=
```
(* one can play with the options here to adapt the embedding *)
coordinates[map_] := get3DGraphEmbedding[uniqueEdges[map], {"SpringElectricalEmbedding", "InferentialDistance" → Automatic, "RepulsiveForcePower" → -2.4}];
```

In[18]:=
```
(* assign colors to faces according to closeness centrality *)
colorFunction[x_] := ColorData["Rainbow"][Round[Max[Min[0.55 - 0.15 x, 1], 0], 1/40]];
centralityColors[map_] := colorFunction /@ Standardize@ClosenessCentrality[mapDualGraph[map]];
```

In[20]:=
```
plotMap3D[map_, coor_] :=
 Graphics3D[{EdgeForm[None], {FaceForm[#[[1]]], Polygon[coor[[#[[2, ;; 3]]]]], If[Length[#[[2]]] == 4, Polygon[coor[[#[[2, {1, 3, 4}]]]]], {}]} & /@
    Transpose[{centralityColors[map], facecycles[map] /. halfedgeToVertexId[map]}]}, Boxed → False]
```

In[21]:=
```
map = generateMap["C", 500];
plotMap3D[map, coordinates[map]]
```

Out[22]=



100%

Menu [Home] mcdt-part1.pdf example-analysi... [Inbox - timothy... 20:05

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

WOLFRAM MATHEMATICA | STUDENT EDITION   Demonstrations | MathWorld | Wolfram Community | Help

```
halfedgeToVertexId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, vertexcycles[map], {2}]];
halfedgeToFaceId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, facecycles[map], {2}]];

In[12]:= (* functions to construct a Mathematica Graph object *)
uniqueEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToVertexId[map])];
uniqueDualEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToFaceId[map])];
mapGraph[map_] := With[{edges = uniqueEdges[map]}, Graph[Union @@ edges, #[[1]] ⟷ #[[2]] & /@ edges, GraphLayout → None]];
mapDualGraph[map_] := With[{edges = uniqueDualEdges[map]}, Graph[Union @@ edges, #[[1]] ⟷ #[[2]] & /@ edges, GraphLayout → None]];
```

Plotting

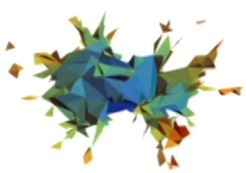
```
In[16]:= (* the following is a bit of a hack to extract coordinates from G
get3DGraphEmbedding[edg_, method_] := (VertexCoordinateRules /. Cases          ing @ DeleteDuplicates[Join @@ (List @@@ edg)]]];

In[17]:= (* one can play with the options here to adapt the embedding *)
coordinates[map_] := get3DGraphEmbedding[uniqueEdges[map], {"SpringEl          }];

In[18]:= (* assign colors to faces according to closeness centrality *)
colorFunction[x_] := ColorData["Rainbow"][Round[Max[Min[0.55 - 0.15 x
centralityColors[map_] := colorFunction /@ Standardize @ ClosenessCentra

In[20]:= plotMap3D[map_, coor_] :=
 Graphics3D[{EdgeForm[None], {FaceForm[#[[1]]], Polygon[coor[[#[[2,
   Transpose[{centralityColors[map], facecycles[map] /. halfedgeToVe

In[21]:= map = generateMap["C", 500];
plotMap3D[map, coordinates[map]]
```

timothy@timothy-UX31A ~

File Edit View Search Terminal Help

```
timothy@timothy-UX31A ~ $
```

Out[22]=



100%

Menu [Home] mcdt-part1.pdf example-analysi... [Inbox - timothy... timothy@timot... 20:05

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

WOLFRAM MATHEMATICA | STUDENT EDITION — Demonstrations | MathWorld | Wolfram Community | Help

Config

```
In[1]:= (* Provide the location of the randgeom executable *)
        (* On windows change randgeom to randgeom.exe *)
        programlocation = FileNameJoin[{NotebookDirectory[], "linux", "randgeom"}]
Out[1]= /home/timothy/Documents/web/homepage/randgeom/linux/randgeom

In[2]:= (* Test the program. If it return False, check the programlocation provided. If it still does not work, first test randgeom from the console. *)
        FileExistsQ[programlocation] && ListQ[RunThrough["'" <> programlocation <> "'", ""]]
Out[2]= True
```

Useful functions

```
In[3]:= (* the following just runs randgeom with the specified parameters and parses the output as Mathematica code *)
        generateMaps[type_, size_, number_] := RunThrough["'" <> programlocation <> "' -t" <> type <> " -s" <> ToString[size] <> " -n" <> ToString[number], ""];
        generateMap[type_, size_] := First@generateMaps[type, size, 1];

In[5]:= (* given a permutation p of {1,2,...,n}, cycles[p] gives the partition of {1,2,...,n} into cycles *)
        cycles[p_] := PermutationCycles[p, Identity];
        (* given a list plist of permutations, orbits[plist] gives the partition of {1,2,...,n} into orbits under the permutations *)
        orbits[plist_] := GroupOrbits@PermutationGroup[PermutationCycles /@ plist]

In[7]:= (* edges, vertices and faces correspond to cycles of halfedge-permutations *)
        edgecycles[map_] := cycles[map[[All, 3]]];
        facecycles[map_] := cycles[map[[All, 1]]];
        vertexcycles[map_] := cycles[map[[map[[All, 3]], 1]]];
        (* We may assign id's to the vertices of map according to their position in vertexcycles[map] *)
        halfedgeToVertexId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, vertexcycles[map], {2}]];
        halfedgeToFaceId[map_] := Dispatch[Join @@ MapIndexed[#1 → #2[[1]] &, facecycles[map], {2}]];

In[12]:= (* functions to construct a Mathematica Graph object *)
        uniqueEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToVertexId[map])];
        uniqueDualEdges[map_] := Union[Sort /@ (edgecycles[map] /. halfedgeToFaceId[map])];
        mapGraph[map_] := With[{edges = uniqueEdges[map]}, Graph[Union @@ edges, #[[1]] ⟷ #[[2]] & /@ edges, GraphLayout → None]];
        mapDualGraph[map_] := With[{edges = uniqueDualEdges[map]}, Graph[Union @@ edges, #[[1]] ⟷ #[[2]] & /@ edges, GraphLayout → None]];
```

Plotting

```
In[16]:= (* the following is a bit of a hack to extract coordinates from GraphPlot3D's embedding of a graph *)
```

100%

Menu | [Home] | mcdt-part1.pdf | example-analysi... | [Inbox - timothy... | [timothy@timot... | 20:06

example-analysis.nb * (Running...) - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

WOLFRAM MATHEMATICA | STUDENT EDITION — Demonstrations | MathWorld | Wolfram Community | Help

```
(* Task: plot geometries of various sizes and models. What are the qualitative differences between the models? *)

(* Task: produce some nice pictures. To save a nice picture, one may use something like Export["picture.png",ImageCrop@Rasterize[plot,ImageSize→800,Background→None]] *)
```

Geodesic two-point function

```
In[23]:= (* Mathematica has built in support for graph distances: this returns a list of distances from all vertices to a randomly chosen vertex *)
distanceListFromRandomPoint[graph_] := GraphDistance[graph, RandomChoice@VertexList[graph]];

In[24]:= (* produce a histogram with the fraction of points at distance 0,1,2,3,... *)
distanceProfile[map_, max_] := BinCounts[#, {0, max}]/Length[#] &@distanceListFromRandomPoint@mapGraph[map];
dualDistanceProfile[map_, max_] := BinCounts[#, {0, max}]/Length[#] &@distanceListFromRandomPoint@mapDualGraph[map];

In[26]:= (* An example of a distance profile (from a random vertex) for a single random geometry *)
ListPlot[distanceProfile[generateMap["C", 500], 30], Joined → True, Filling → Axis, InterpolationOrder → 0, AxesLabel → {"r", "ρ(r)"}]

(* the same but for dual graph distance *)
ListPlot[dualDistanceProfile[generateMap["C", 500], 30], Joined → True, Filling → Axis, InterpolationOrder → 0, AxesLabel → {"r", "ρ(r)"}]

(* Task: Proceed to gather data for the average distance profile for different system sizes and models, and attempt finite-size scaling to extract the Hausdorff dimension *)
```

Spectral dimension

The spectral dimension d_s of a map is related to the probability $p(t)$ that a simple random walk on the map (or its dual) returns to its starting point after t steps: $p(t) \sim t^{-d_s/2}$ for 1<< t << n (where n is the system size). There are various ways to measure this return probability: one can simulate a random walker and just record its returns; study a heat diffusion process; or use linear algebra as follows. The return probability ⟨p(t)⟩ averaged over all starting points of the map is related to the normalized adjacency matrix A via $\langle p(t)\rangle = \mathrm{Tr}(A^t)/n = \sum \lambda_i{}^t / n$, where λ_i are the eigenvalues of A.

```
dualAdjacencyMatrix[map_] := Module[{dualedges, matrixentries, facedegrees},
  dualedges = edgecycles[map] /. halfedgeToFaceId[map];
  facedegrees = Length /@ facecycles[map];
  matrixentries = Tally@Join[dualedges, Reverse /@ dualedges];
  SparseArray[#[[1]] → #[[2]]/facedegrees[[#[[1, 1]]]] & /@ matrixentries]
 ]

(* plot the adjacency matrix (for fun) *)
MatrixPlot[dualAdjacencyMatrix[generateMap["D", 300]]]
```

Menu | [Home] | mcdt-part1.pdf | example-analysi... | [Inbox - timothy... | [timothy@timot... | 20:07 | 100%

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

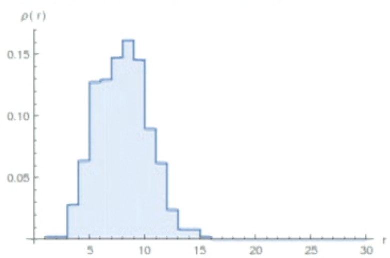
WOLFRAM MATHEMATICA | STUDENT EDITION   Demonstrations | MathWorld | Wolfram Community | Help

```
(* Task: plot geometries of various sizes and models. What are the qualitative differences between the models? *)

(* Task: produce some nice pictures. To save a nice picture, one may use something like Export["picture.png",ImageCrop@Rasterize[plot,ImageSize→800,Background→None]] *)
```

Geodesic two-point function

```
In[23]:= (* Mathematica has built in support for graph distances: this returns a list of distances from all vertices to a randomly chosen vertex *)
distanceListFromRandomPoint[graph_] := GraphDistance[graph, RandomChoice@VertexList[graph]];

In[24]:= (* produce a histogram with the fraction of points at distance 0,1,2,3,... *)
distanceProfile[map_, max_] := BinCounts[#, {0, max}] / Length[#] &@distanceListFromRandomPoint@mapGraph[map];
dualDistanceProfile[map_, max_] := BinCounts[#, {0, max}] / Length[#] &@distanceListFromRandomPoint@mapDualGraph[map];

In[26]:= (* An example of a distance profile (from a random vertex) for a single random geometry *)
ListPlot[distanceProfile[generateMap["C", 500], 30], Joined → True, Filling → Axis, InterpolationOrder → 0, AxesLabel → {"r", "ρ(r)"}]
```

Out[26]=




```
(* the same but for dual graph distance *)
ListPlot[dualDistanceProfile[generateMap["C", 500], 30], Joined → True, Filling → Axis, InterpolationOrder → 0, AxesLabel → {"r", "ρ(r)"}]

(* Task: Proceed to gather data for the average distance profile for different system sizes and models, and attempt finite-size scaling to extract the Hausdorff dimension *)
```

Spectral dimension

The spectral dimension d_s of a map is related to the probability $p(t)$ that a simple random walk on the map (or its dual) returns to its starting point after t steps: $p(t) \sim t^{-d_s/2}$ for 1<< t << n (where n is the system size). There are various ways to measure this return probability: one can simulate a random walker and just record its returns; study a heat diffusion process; or use linear algebra as follows. The return probability ⟨p(t)⟩ averaged over all starting points of the map is related to the normalized adjacency matrix

Menu [Home] mcdt-part1.pdf example-analysi... [Inbox - timothy... [timothy@timot... 20:07 100%

example-analysis.nb * - Wolfram Mathematica 11.0 Student Edition - Personal Use Only

File Edit Insert Format Cell Graphics Evaluation Palettes Window Help

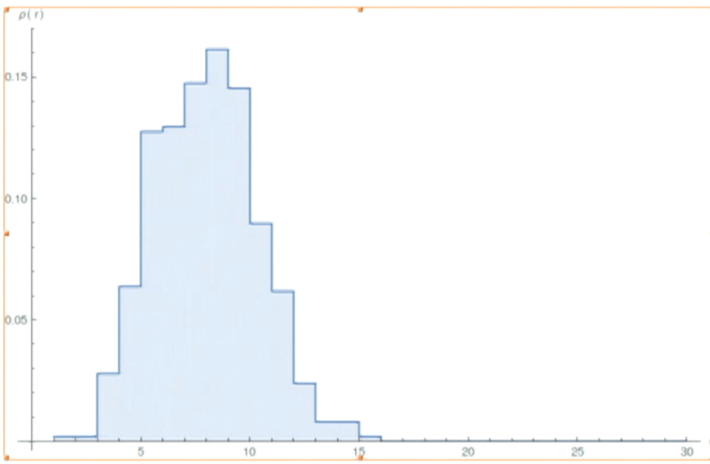
WOLFRAM MATHEMATICA | STUDENT EDITION — Demonstrations | MathWorld | Wolfram Community | Help

```
(* Task: plot geometries of various sizes and models. What are the qualitative differences between the models? *)

(* Task: produce some nice pictures. To save a nice picture, one may use something like Export["picture.png",ImageCrop@Rasterize[plot,ImageSize→800,Background→None]] *)
```

Geodesic two-point function

```
In[23]:= (* Mathematica has built in support for graph distances: this returns a list of distances from all vertices to a randomly chosen vertex *)
distanceListFromRandomPoint[graph_] := GraphDistance[graph, RandomChoice@VertexList[graph]];

In[24]:= (* produce a histogram with the fraction of points at distance 0,1,2,3,... *)
distanceProfile[map_, max_] := BinCounts[#, {0, max}]/Length[#] &@distanceListFromRandomPoint@mapGraph[map];
dualDistanceProfile[map_, max_] := BinCounts[#, {0, max}]/Length[#] &@distanceListFromRandomPoint@mapDualGraph[map];

In[26]:= (* An example of a distance profile (from a random vertex) for a single random geometry *)
ListPlot[distanceProfile[generateMap["C", 500], 30], Joined → True, Filling → Axis, InterpolationOrder → 0, AxesLabel → {"r", "ρ(r)"}]
```

Out[26]=



100%

Menu [Home] mcdt-part1.pdf example-analysi... [Inbox - timothy... [timothy@timot... 20:07

```
g++ main.cpp -I. -O2 -o randgeom
```

Usage of `randgeom`

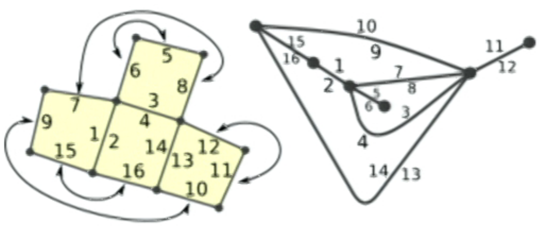
`randgeom` takes three parameters:

- `-t` followed by A, B, C, or D: the requested model type.
- `-s` followed by positive integer S: the requested size S of the planar map measured by number of faces.
- `-n` followed by positive integer N: the number N of independent configurations to be returned.

The following returns a single random planar map sampled from model A with 4 faces.

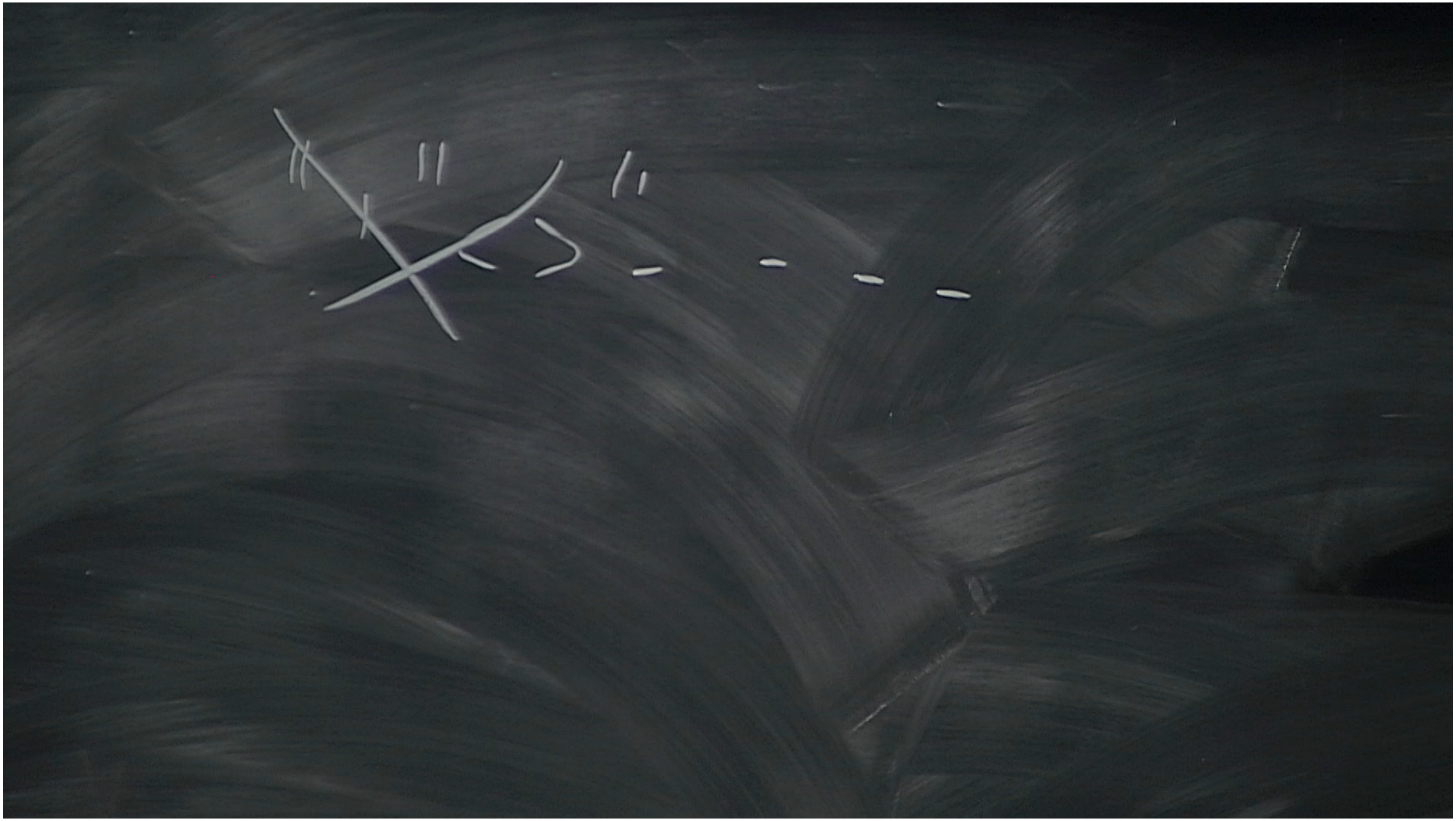
```
$ ./randgeom -tA -s4 -n1
{{{7,15,2},{16,4,1},{8,6,4},{2,14,3},{6,8,6},{3,5,5},{9,1,8},{5,3,7},{15,7,10},{11,13,9},{12,10,12},
{13,11,11},{10,12,14},{4,16,13},{1,9,16},{14,2,15}}}
```

The output is formatted as a Mathematica-style nested list of length N, one entry per configuration. Each configuration corresponds to a list of triples of integers describing a planar map through permutations: the `i`'th triple corresponds to $\{n(i), n^{-1}(i), a(i)\}$, where `n` and `a` are the ``next'' and ``adjacent'' permutations (see lecture slides). The output above corresponds to the following quadrangulation displayed both as a gluing prescription and as a planar map.



In case you prefer to read the output from a different program than Mathematica, you may prefer to receive the data as a space-separated list. To this end one may use the option `--spaceseparated`:

```
$ ./randgeom -tA -s2 -n2 --spaceseparated
2
8
3 7 2
8 4 1
5 1 4
2 6 3
7 3 6
4 8 5
1 5 8
```

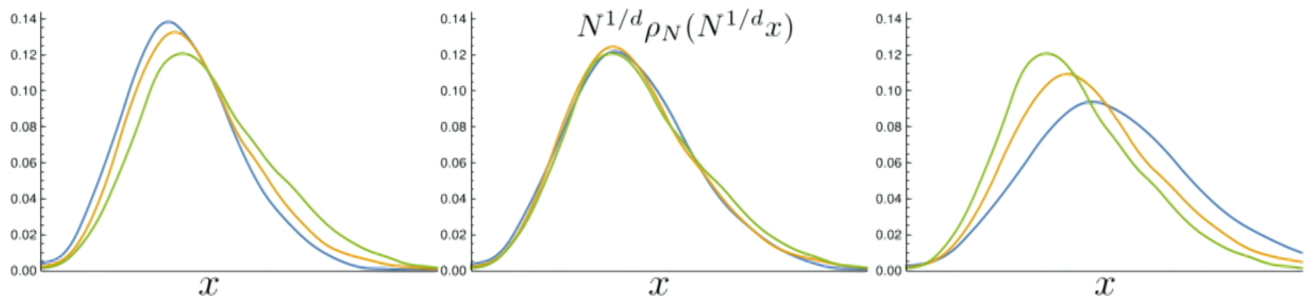


Observables: geodesic distances (continued)

- Instead of taking $N \to \infty$ and then $r \to \infty$, it is usually better to use *finite-size scaling methods*: one expects

$$N^{1/d_{\mathrm{H}}} \rho_N(N^{1/d_{\mathrm{H}}} x) \text{ to converge as } N \to \infty \text{ for any fixed } x \in \mathbb{R}.$$

- Equivalently, we expect the distribution of the distance between two random points to converge as $N \to \infty$ provided we take edge lengths $\sim N^{-1/d_{\mathrm{H}}}$.
- Estimate d_H by "collapsing curves":



localhost:8888/notebooks/AppData/Local/Julia-0.5.2/Quantum-Cuboid-renormalization.ipynb

jupyter Quantum-Cuboid-renormalization Last Checkpoint: 8 hours ago (autosaved)

File Edit View Insert Cell Kernel Help — Julia 0.5.0

Tutorial: Quantum cuboid renormalisation

Basic definitions

In [ ]:

```
function DE(j1::Float64,j2::Float64,j3::Float64,j4::Float64,j5::Float64,j6::Float64)
    res = (-2) * (j1^2 * (j2 + j4) + j2 * j4 * (j2 + j4) + j1 * (j2^2 + (1 + 1im) * j2 * j4 + j4^2))
    res *= (j1^2 * (j3 + j5) + j3 * j5 * (j3 + j5) + j1 * (j3^2 + (1 + 1im) * j3 * j5 + j5^2))
    res *= (j3 * j4 * j5 + j2 * (j4 * j5 + j3 * (j4 + j5)))
    res *= (j2^2 * (j3 + j6) + j3 * j6 * (j3 + j6) + j2 * (j3^2 + (1 + 1im) * j3 * j6 + j6^2))
    res *= (j4^2 * (j5 + j6) + j5 * j6 * (j5 + j6) + j4 * (j5^2 + (1 + 1im) * j5 * j6 + j6^2))
    res *= (j3 * j4 * j6 + j1 * (j4 * j6 + j3 * (j4 + j6)))
    res *= (j2 * j5 * j6 + j1 * (j5 * j6 + j2 * (j5 + j6)))

    res
end

function DF(j1::Float64,j2::Float64,j3::Float64,j4::Float64,j5::Float64,j6::Float64)
    res = (-2) * (j1^2 * (j2 + j4) + j2 * j4 * (j2 + j4) + j1 * (j2^2 + (1 - 1im) * j2 * j4 + j4^2))
    res *= (j1^2 * (j3 + j5) + j3 * j5 * (j3 + j5) + j1 * (j3^2 + (1 - 1im) * j3 * j5 + j5^2))
    res *= (j3 * j4 * j5 + j2 * (j4 * j5 + j3 * (j4 + j5)))
    res *= (j2^2 * (j3 + j6) + j3 * j6 * (j3 + j6) + j2 * (j3^2 + (1 - 1im) * j3 * j6 + j6^2))
    res *= (j4^2 * (j5 + j6) + j5 * j6 * (j5 + j6) + j4 * (j5^2 + (1 - 1im) * j5 * j6 + j6^2))
    res *= (j3 * j4 * j6 + j1 * (j4 * j6 + j3 * (j4 + j6)))
    res *= (j2 * j5 * j6 + j1 * (j5 * j6 + j2 * (j5 + j6)))

    res
end

function Ampl(α::Float64,j1::Float64,j2::Float64,j3::Float64,j4::Float64,j5::Float64,j6::Float64)
    res = 1/((2. * pi)^3) * 2.^(24)
    res *= (1/(sqrt(-DE(j1,j2,j3,j4,j5,j6))) +1/(sqrt(-DF(j1,j2,j3,j4,j5,j6))))
    res *= (1/(sqrt(-DE(j1,j2,j3,j4,j5,j6))) + 1/(sqrt(-DF(j1,j2,j3,j4,j5,j6))))
```