Mathematica code for the simulations of the paper: "Scaling limits of planar maps under the Smith embedding"

Federico Bertacco
Imperial College London

Ewain Gwynne
University of Chicago

Scott Sheffield
MIT

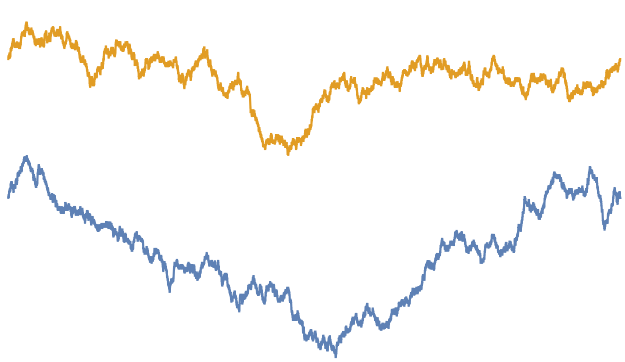
Abstract

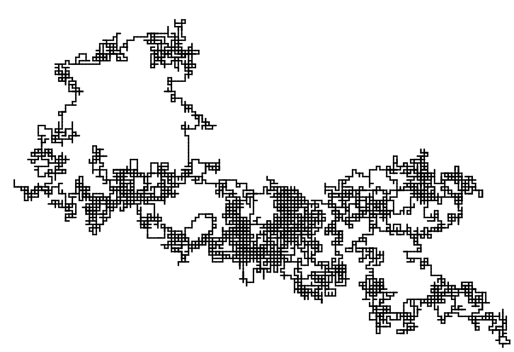
In this document, we provide the Mathematica code used to produce the simulations of the Smith embedding of a planar map contained in the paper "Scaling limits of planar maps under the Smith embedding".

1 Code and outputs

The Mathematica code below generates a random walk (X, Y) in $\mathbb{Z}^2$ conditioned to end where it started after n steps. The code starts by computing (A, B), which is the 45-degree-rotated version of (X, Y) whose coordinates are independent one-dimensional walks conditioned to end at zero. It then displays two figures, one showing the trace of the walk in $\mathbb{Z}^2$ and the other plotting the X and Y coordinates separately as functions of time.

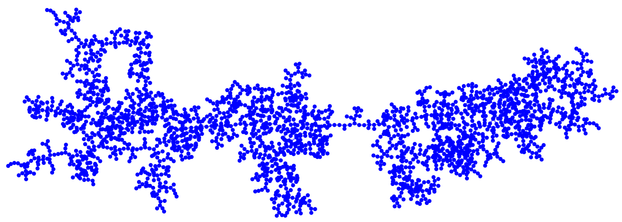
```
n = 2000;
Z = Table[0, {j, 1, n + 1}];
A = Z;
B = Z;

For[j = 1, j < n, ++j,
  A[[j + 1]] = A[[j]] + If[2 RandomReal[] > 1 + A[[j]]/(n - j), 1, -1];
  B[[j + 1]] = B[[j]] + If[2 RandomReal[] > 1 + B[[j]]/(n - j), 1, -1]
];

X = n/2 + (A + B)/2;
Y = n/2 + (A - B)/2;

{
  ListPlot[{X, n + Sqrt[n] - Y}, PlotJoined -> True, Axes -> False],
  Graphics[
    Table[Line[{{X[[j]], Y[[j]]}, {X[[j + 1]], Y[[j + 1]]}}], {j, 1, n}]
  ]
}
```

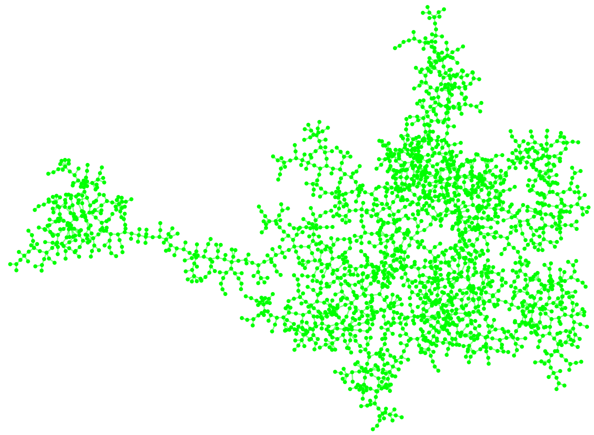




The graph of X determines a tree obtained by identifying points connected by a chord under the graph, with the two endpoints of $[0, n]$ identified so that chords can wrap around. An embedding of such a tree is shown below in blue (with a similar tree for Y shown in green).

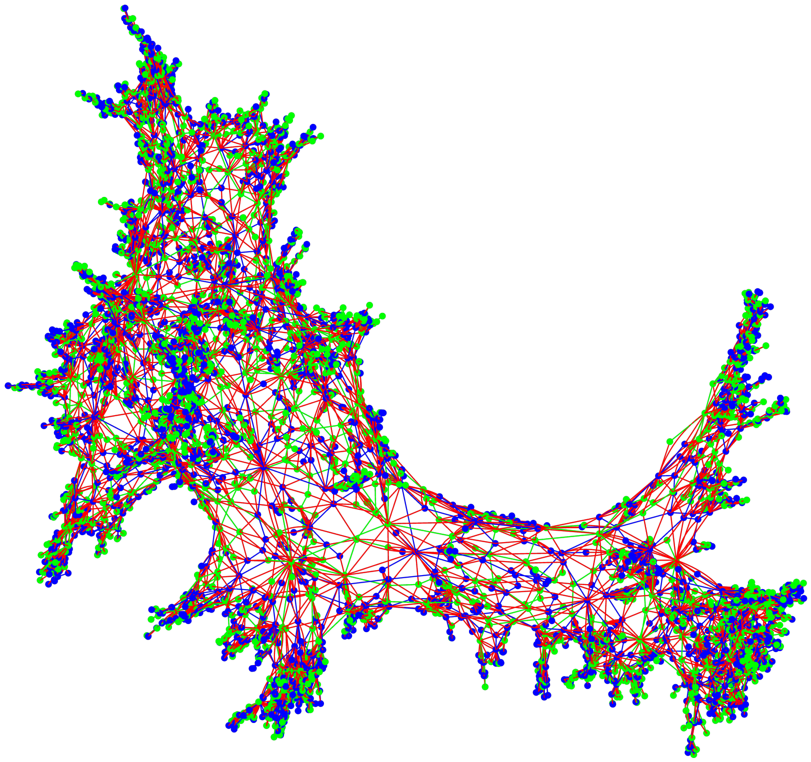
```
vertnumX = Z + 1;
vertnumY = Z + 1;
last = Z + 1;
minxloc = 1;
minyloc = 1;

For[j = 1, j < n + 1, ++j,
  If[X[[j]] < X[[minxloc]], minxloc = j];
  If[Y[[j]] < Y[[minyloc]], minyloc = j]
];

count = 1;

For[j = minxloc, j < n + 1, ++j,
  If[X[[j + 1]] > X[[j]],
    vertnumX[[j + 1]] = ++count;
    last[[X[[j + 1]]]] = count,
    vertnumX[[j + 1]] = last[[X[[j + 1]]]]
  ]
];

vertnumX[[1]] = vertnumX[[n + 1]];

For[j = 1, j < minxloc - 1, ++j,
  If[X[[j + 1]] > X[[j]],
    vertnumX[[j + 1]] = ++count;
    last[[X[[j + 1]]]] = count,
    vertnumX[[j + 1]] = last[[X[[j + 1]]]]
  ]
];

vertnumY[[minyloc]] = ++count;
last = Z + count;

For[j = minyloc, j < n + 1, ++j,
  If[Y[[j + 1]] > Y[[j]],
    vertnumY[[j + 1]] = ++count;
    last[[Y[[j + 1]]]] = count,
    vertnumY[[j + 1]] = last[[Y[[j + 1]]]]
  ]
];

vertnumY[[1]] = vertnumY[[n + 1]];

For[j = 1, j < minyloc - 1, ++j,
  If[Y[[j + 1]] > Y[[j]],
    vertnumY[[j + 1]] = ++count;
    last[[Y[[j + 1]]]] = count,
    vertnumY[[j + 1]] = last[[Y[[j + 1]]]]
  ]
];

{
  GraphPlot[
    SimpleGraph[Table[Style[vertnumX[[j]] <-> vertnumX[[j + 1]], Blue], {j, 1, n}]],
    VertexStyle -> Blue,
    GraphLayout -> {"SpringEmbedding"}
  ],
  GraphPlot[
    SimpleGraph[Table[Style[vertnumY[[j]] <-> vertnumY[[j + 1]], Green], {j, 1, n}]],
    VertexStyle -> Green,
    GraphLayout -> {"SpringEmbedding"}
  ]
}
```





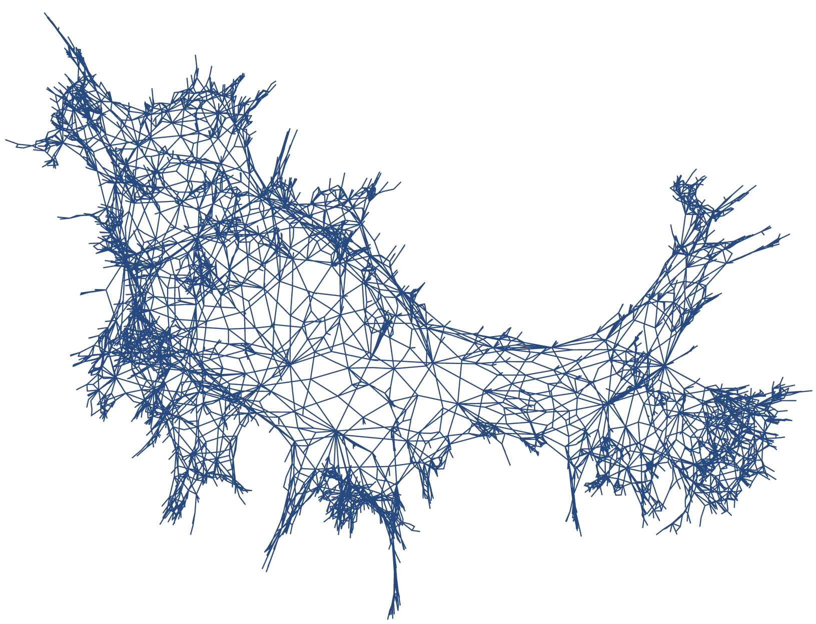
We can then draw in the red edges between the blue and green trees to construct and illustrate a graph object as follows.

```
g = Table[0 -> 0, {j, 1, 3 n/2}];
count = 1;

For[j = 1, j <= n, ++j,
  g[[count++]] = Style[vertnumX[[j]] <-> vertnumY[[j]], Red]
];

For[j = 1, j <= n, ++j,
  If[X[[j]] < X[[j + 1]],
    g[[count++]] = Style[vertnumX[[j]] <-> vertnumX[[j + 1]], Blue]
  ]
];

For[j = 1, j <= n, ++j,
  If[Y[[j]] < Y[[j + 1]],
    g[[count++]] = Style[vertnumY[[j]] <-> vertnumY[[j + 1]], Green]
  ]
];

GraphPlot3D[
  g,
  VertexStyle -> Table[i -> If[i < vertnumY[[minyloc]], Blue, Green], {i, 1, n/2 + 2}],
  GraphLayout -> {"SpringElectricalEmbedding"}
]
```



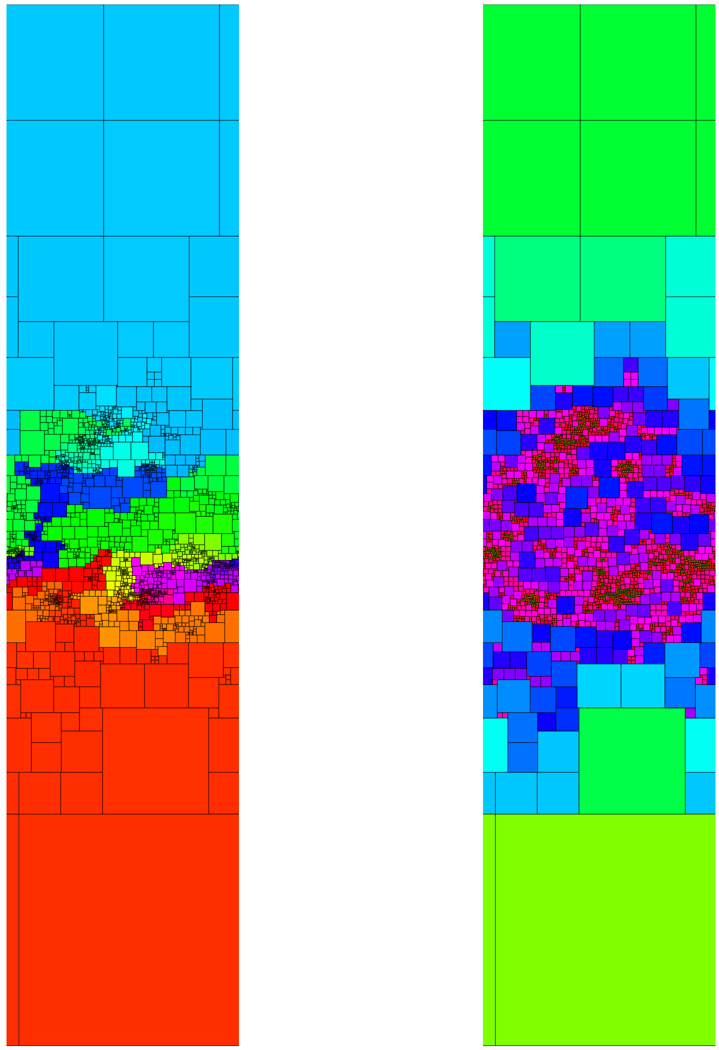
Alternatively, one can represent the graph using a sparse array M, which is more efficient when the number of vertices is large. The code below displays only the red edges, so the map is a quadrangulation.

```
M = SparseArray[Table[0, {j, 1, n/2 + 2}, {k, 1, n/2 + 2}]];
For[j = 1, j < n + 1, j++,
  M[[vertnumX[[j]], vertnumY[[j]]]] += 1
];
x = GraphPlot3D[M,
  GraphLayout -> {"SpringElectricalEmbedding"},
  VertexShapeFunction -> None
]
```



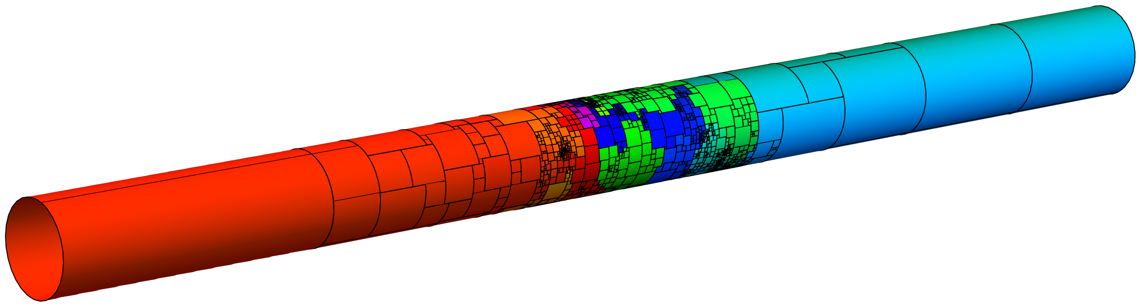
Once M is computed as above we can use some linear algebra to compute a function w that is discrete harmonic except at the points a and b corresponding to the root and dual root of the two trees. We can then display the corresponding Smith embedding by positioning the squares one by one. The functions `sq` and `ssq` give the two different color schemes, one by size, one by order w.r.t. the space-filling path that snakes between the tree and dual tree.

```
M = M + Transpose[M];
deg = Table[Sum[M[[i, j]], {i, 1, n/2 + 2}], {j, 1, n/2 + 2}];
L = Table[If[i == j, -deg[[i]], M[[i, j]]], {i, 1, n/2 + 2}, {j, 1, n/2 + 2}];
a = vertnumX[[minxloc]];
b = vertnumY[[minyloc]];

For[j = 1, j <= n/2 + 2, ++j,
  L[[a, j]] = 0;
  L[[b, j]] = 0;
];
L[[a, a]] = 1;
L[[b, b]] = 1;

v = Table[0, {j, 1, n/2 + 2}];
v[[a]] = 1;
w = LinearSolve[N[L], N[v]];

horiz = Table[0, {j, 1, n + 1}];
vertgap = Table[w[[vertnumY[[j]]]] - w[[vertnumX[[j]]]], {j, 1, n + 1}];
horiz[[1]] = 0;

For[j = 1, j <= n, ++j,
  horiz[[j + 1]] = horiz[[j]] + vertgap[[j]];
];

horizgap = Abs[horiz[[n + 1]]];
g = Table[0, {j, 1, n}];
count = 1;

sq[bot_, top_, left_, hue_] := {Hue[hue], EdgeForm[Thin], Rectangle[{left, bot}, {left + (
    top - bot), top}]};
ssq[bot_, top_, left_, hue_] := {Hue[-Log[Abs[top - bot] + .0000001]/6], EdgeForm[Thin],
    Rectangle[{left, bot}, {left + (top - bot), top}]};

g1 = Table[sq[w[[vertnumX[[j]]]], w[[vertnumY[[j]]]], horiz[[j]], j/n], {j, 1, n}];
g2 = Table[ssq[w[[vertnumX[[j]]]], w[[vertnumY[[j]]]], horiz[[j]], j/n], {j, 1, n}];

{
  Graphics[{g1, Translate[g1, {horizgap, 0}], Translate[g1, {2 horizgap, 0}]}, PlotRange ->
    {{0, horizgap}, {0, 1}}],
  Graphics[{g2, Translate[g2, {horizgap, 0}], Translate[g2, {2 horizgap, 0}]}, PlotRange ->
    {{0, horizgap}, {0, 1}}]
}
```



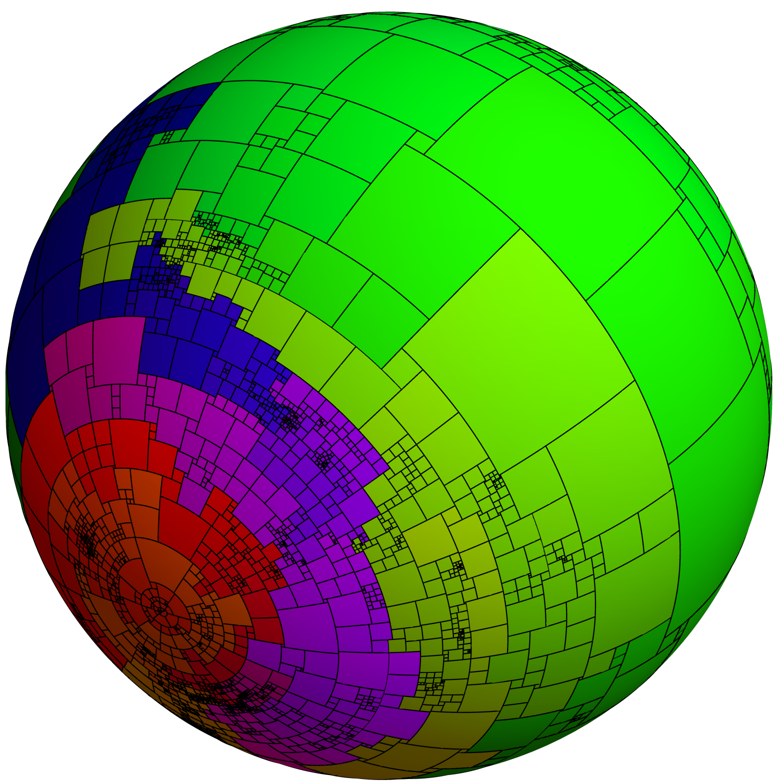
Replacing the square-embedding function `sq` with a cylinder-embedding function `rsvq` gives an embedding in the cylinder. Here we only display squares of size about some `cutoff` so that computational time is not wasted on drawing squares that may be too small to see.

```
cutoff = .0001;
rvsq[bot_, top_, left_, hue_] := RevolutionPlot3D[
{1, \[Theta]},
{\[Theta], (2 Pi/horizgap) bot, (2 Pi/horizgap) top + .0000001},
{p, (2 Pi/horizgap) left, (2 Pi/horizgap) (left + (top - bot) + .0000001)},
Mesh -> None,
PlotStyle -> Hue[hue],
BoundaryStyle -> {None, Black}
];

count = 0;
r = Table[0, {j, 1, n}];

For[j = 1, j <= n, ++j,
If[Abs[w[[vertnumX[[j]]]] - w[[vertnumY[[j]]]]] > .0001,
r[[++count]] = rvsq[w[[vertnumX[[j]]]], w[[vertnumY[[j]]]], horiz[[j]], j/n]
];
];

Show[Table[r[[j]], {j, 1, count}], PlotRange -> All, Boxed -> False, Axes -> False]
```



A similar function **spsq** (using a Mercator projection) gives an embedding in the sphere instead of cylinder.

```
cutoff = .0001;

spsq[bot_, top_, left_, hue_] := SphericalPlot3D[
  1,
  {p, 2 ArcTan[Exp[(2 Pi/horizgap) (bot - 1/2)]], 2 ArcTan[Exp[(2 Pi/horizgap) (top - 1/2)]
    + .0000001]},
  {\[Theta], (2 Pi/horizgap) left, (2 Pi/horizgap) (left + (top - bot)) + .0000001},
  Mesh -> None,
  PlotStyle -> Hue[hue],
  BoundaryStyle -> {None, Black}
];

count = 0;
For[j = 1, j <= n, ++j,
  If[Abs[w[[vertnumX[[j]]]] - w[[vertnumY[[j]]]]] > .0001,
    r[[++count]] = spsq[w[[vertnumX[[j]]]], w[[vertnumY[[j]]]], horiz[[j]], j/n]
  ];
];

Show[Table[r[[j]], {j, 1, count}], PlotRange -> All, Boxed -> False, Axes -> False]
```



We remark that one can also type the following after any of the 3D figures to export an animation showing a rotating version of the image.

```
ResourceFunction["ExportRotatingGIF"]["C:\\filename.gif", %, ImageSize -> 1200]
```