

11.2 Spreading of Fire

File: *Fire.nb*

Introduction to Computational Science: Modeling and Simulation for the Sciences
Angela B. Shiflet and George W. Shiflet
Wofford College
© 2006 by Princeton University Press

Based on Gaylord, Richard J., and Kazume Nishidate. 1996. "Contagion in Excitable Media." *Modeling Nature: Cellular Automata Simulations with Mathematica*. New York; TELOS/Springer-Verlag: 155-171.

Development of Fire Program

Lattice (matrix, grid)

Grid-site values:

EMPTY (0) - empty
TREE (1) - non-burning tree
BURNING (2) - burning tree

probTree = probability of grid site occupied by tree (value 1); i.e., tree density

probBurning = probability that a tree is burning (value 2); i.e., fraction of burning trees

probImmune = probability of immunity from catching fire - global variable

probLightning = probability of lightning - global variable

```
(* Constants *)  
EMPTY = 0;  
TREE = 1;  
BURNING = 2;
```

Update rules

Rules to compute next site value

- At next time step an empty site remains empty

```
spread[EMPTY, _, _, _, _] := EMPTY;
```

- Burning tree results in empty cell next time step

```
spread[BURNING, _, _, _, _] := EMPTY;
```

- Perhaps next time step tree with burning neighbor(s) burns itself

Variable *probImmune* must be undefined for rule definitions but must be given a value before rules are used.

```
Clear[probImmune]
spread[TREE, N_, E_, S_, W_] :=
  If[RandomReal[] < probImmune, TREE, BURNING] /;
  MatchQ[BURNING, N | E | S | W];
```

- Perhaps tree is hit by lightning and burns next time step

Variables *probImmune* and *probLightning* must be undefined for rule definitions but must be given values before rules are used.

```
Clear[probImmune, probLightning]
spread[TREE, _, _, _, _] :=
  If[RandomReal[] < probLightning * (1 - probImmune), BURNING, TREE];
```

Boundaries

Periodic boundary conditions in this case

Infinite system

Extend boundary by 1 for *sniff* and by 2 for *walk*

■ Function to return an $(n + 2)$ -by- $(n + 2)$ matrix for periodic boundaries *mat*, n -by- n matrix

```

Clear[extendLat]
extendLat[mat_] := Module[{matNS, trans, transEW},
  (* glue on wrap of N & S rows *)
  matNS = Join[Take[mat, -1], mat, Take[mat, 1]];

  (* glue on wrap of E & W columns assuming original *)
  (* matrix is square *)
  trans = Transpose[matNS];
  transEW = Join[Take[trans, -1], trans, Take[trans, 1]];

  Transpose[transEW]
];

```

Function to apply a function parameter to extended matrix

■ Function to apply a function parameter to a matrix extended by 1 cell in each direction

Function returns a matrix of applications of function *fnc* (*fnc[site, N, E, S, W]*) to each element (*site*) of extended matrix *matExt* except for the first and last rows and columns

```

applyExtended[fnc_, matExt_] := Module[{n, site, N, E, S, W, i, j},
  n = Length[matExt] - 2;

  Table[
    site = matExt[[i, j]];
    N = matExt[[i - 1, j]];
    E = matExt[[i, j + 1]];
    S = matExt[[i + 1, j]];
    W = matExt[[i, j - 1]];
    fnc[site, N, E, S, W],

    {i, 2, n + 1}, {j, 2, n + 1}
  ];

```

Fire simulation

- Function *fire* drives the simulation and returns a list of grid matrices.

Grid-site values:

EMPTY (0) - empty

TREE (1) - non-burning tree

BURNING (2) - burning tree

Variables *probLightning* and *probImmune* must global for spread rules.

n = number of cells in each direction

probTree = probability of grid site occupied by tree; i.e., tree density

probBurning = probability that a tree is burning; i.e., fraction of burning trees

chanceLightning (globally *probLightning*) = probability of lightning

chanceImmune (globally *probImmune*) = probability of immunity from catching fire

t = number of time steps in simulation

```
Clear[fire]
fire[n_, probTree_, probBurning_, chanceLightning_, chanceImmune_,
  t_] :=
Module[{forest, forestExtended},

  (** Assign values to global variables **)
  probLightning = chanceLightning;
  probImmune = chanceImmune;

  (** Initialize grid **)
  forest = Table[
    If[RandomReal[] < probTree,
      If[RandomReal[] < probBurning, BURNING, TREE],
      EMPTY],
    {n}, {n}];

  (** Perform simulation **)
  grids = {forest};
  Do[
    (* Extend matrix *)
    forestExtended = extendLat[forest];

    (* Apply spread of fire function to each grid point *)
    forest = applyExtended[spread, forestExtended];

    (* Save new matrix *)
    grids = Append[grids, forest],
    {t}
  ];

  grids
]
```

Animation of simulation

- Function to display a list of grids with cell values colored as follows: *EMPTY* → yellow, *TREE* → green, *BURNING* → burnt orange

```

Clear[showGraphs]
(* Function to display a list of grids starting at 1 through *)
(* entire list of grids *)
(* Empty (0) shows yellow; tree (1) shows green; *)
(* burning tree (2) shows burnt orange. *)

showGraphs[graphList_] := Module[{k, g},
  rasterList = {};
  Do[
    g = graphList[[k]]; (* grid at k-th time step *)
    AppendTo[rasterList,
      Graphics[Raster[Reverse[g]] /.
        {EMPTY → {1, 1, 0}, TREE → {0.1, 0.75, 0.2},
        BURNING → {0.6, 0.2, 0.1}}]],
    {k, Length[graphList]}
  ];
  ListAnimate[rasterList]
]

```

Run Program

fire[*n*, *probTree*, *probBurning*, *probLightning*, *probImmune*, *t*]

- Initialization with $n = 20$, $probTree = 0.5$, $probBurning = 0.5$

```
fireList = fire[20, 0.5, 0.5, 0.5, 0.5, 5];
```

```
showGraphs[fireList]
```

■ *probLightning* = 0*probImmune* = 0

```
fireList = fire[20, 0.3, 0.001, 0, 0, 15];
```

```
showGraphs[fireList]
```

■ *probLightning* small*probImmune* = 0

```
fireList = fire[20, 0.3, 0.05, 0.00025, 0, 20];
```

```
showGraphs[fireList]
```

■ *probLightning* = 0*probImmune* = 0.52

```
forestFire[20, 0.3, 0.2, 0, 0.52, 25]
```

```
showGraphs[fireList]
```

Fire Functions Together

```

Clear[spread, extendLat, applyExtended, fire, showGraphs]

(* Constants *)
EMPTY = 0;
TREE = 1;
BURNING = 2;

(***** Spreading Rules *****)
(* spread[site, N, E, S, W] *)

(* At next time step an empty site remains empty *)
spread[EMPTY, _, _, _, _] := EMPTY;

(* Burning tree results in empty cell next time step *)
spread[BURNING, _, _, _, _] := EMPTY;

(* Perhaps next time step tree with burning neighbor(s) burns
itself *)
(* Variable probImmune must be undefined for rule definitions but *)
(* must be given a value before rules are used. *)
Clear[probImmune];
spread[TREE, N_, E_, S_, W_] :=
  If[RandomReal[] < probImmune, TREE, BURNING] /;
  MatchQ[BURNING, N | E | S | W];

(* Perhaps tree is hit by lightning and burns next time step *)
(* Variables probImmune and probLightning must be undefined *)
(* for rule definitions but must be given values before rules *)
(* are used. *)

Clear[probImmune, probLightning]
spread[TREE, _, _, _, _] :=
  If[RandomReal[] < probLightning * (1 - probImmune), BURNING, TREE];

```

```
(*****
(* Function to return an (n + 2)-by-(n + 2) matrix *)
(* with periodic boundaries for mat, an n-by-n matrix *)

extendLat[mat_] := Module[{matNS, trans, transEW},
  (* glue on wrap of N & S rows *)
  matNS = Join[Take[mat, -1], mat, Take[mat, 1]];

  (* glue on wrap of E & W columns assuming original matrix
  is square *)
  trans = Transpose[matNS];
  transEW = Join[Take[trans, -1], trans, Take[trans, 1]];

  Transpose[transEW]
];
```

```
(* Function to return a matrix of applications of function fnc *)
(* (fnc[site, N, E, S, W]) to each element (site) of extended
matrix *)
(* matExt except for the first and last rows and columns *)

applyExtended[fnc_, matExt_] := Module[{n, site, N, E, S, W, i, j},
  n = Length[matExt] - 2;

  Table[
    site = matExt[[i, j]];
    N = matExt[[i - 1, j]];
    E = matExt[[i, j + 1]];
    S = matExt[[i + 1, j]];
    W = matExt[[i, j - 1]];
    fnc[site, N, E, S, W],

    {i, 2, n + 1}, {j, 2, n + 1}
  ];
```

```
(*****
(* Function returns list of grids simulating fire spread *)
(* In site, *)
(*   EMPTY (0) - empty, *)
(*   TREE (1) - non-burning tree, *)
(*   BURNING (2) - burning tree *)
(* Next value based on site and nearest neighbors (N, E, S, W) *)
(* n-by-n grid, periodic boundary conditions *)
```

```

(* probTree - probability of tree in site *)
(* probBurning - probability of tree burning *)
(* chanceLightning(globally probLightning)
- probability of lightning *)
(* chanceImmune(globally probImmune) - probability of immunity *)
(*           from catching fire *)
(* spread - function for updating rules *)
(* t - number of time steps *)

fire[n_, probTree_, probBurning_, chanceLightning_, chanceImmune_,
t_] :=
Module[{forest, forestExtended},

  (** Assign values to global variables **)
  probLightning = chanceLightning;
  probImmune = chanceImmune;

  (** Initialize grid **)
  forest = Table[
    If[RandomReal[] < probTree,
      If[RandomReal[] < probBurning, BURNING, TREE],
      EMPTY],
    {n}, {n}];

  (** Perform simulation **)
  grids = {forest};
  Do[
    (* Extend matrix *)
    forestExtended = extendLat[forest];

    (* Apply spread of fire function to each grid point *)
    forest = applyExtended[spread, forestExtended];

    (* Save new matrix *)
    grids = Append[grids, forest],
    {t}
  ];

  grids
]

```

```
(*****)  
(* Function to display a list of grids starting at 1 through  
entire list of grids *)  
(* Empty (EMPTY = 0) shows yellow; tree (TREE = 1) shows green; *)  
(* burning tree (BURNING = 2) shows burnt orange. *)  
  
Clear[showGraphs]  
showGraphs[graphList_] := Module[{k, g},  
  rasterList = {};  
  Do[  
    g = graphList[[k]]; (* grid at k-th time step *)  
    AppendTo[rasterList,  
      Graphics[Raster[Reverse[g]] /.  
        {EMPTY → {1, 1, 0}, TREE → {0.1, 0.75, 0.2},  
          BURNING → {0.6, 0.2, 0.1}}]],  
    {k, Length[graphList]}  
  ];  
  ListAnimate[rasterList]  
]
```