Supplementary information for "Creative computing with Landlab: an open-source toolkit for building, coupling, and exploring two-dimensional numerical models of earth-surface dynamics", by Daniel E. J. Hobley, Jordan M. Adams, Sai Siddhartha Nudurupati, Eric W. H. Hutton, Nicole M. Gasparini, Erkan Istanbulluoglu, Gregory E. Tucker

This document contains:

Table S1: *A list of all Landlab standard names in Landlab v.1.0.*

Scripts S2-S6: *Scripts to run the models described in section 5 of the main text. Note S5 also contains the text of a separate input file after the code block.*

Table S1. Field names used by Landlab version 1.0.

Field name	Used by	Provided by
channel__bed_shear_stress		SedDepEroder
channel__chi_index		ChiFinder
channel__depth		SedDepEroder
channel__discharge		SedDepEroder
channel__steepness_index		SteepnessFinder
channel__width		SedDepEroder
channel_sediment__relative_flux		SedDepEroder
channel_sediment__volumetric_flux		SedDepEroder
channel_sediment__volumetric_transport_capacity		SedDepEroder
depression__depth	DepressionFinderAndRouter	
depression__outlet_node	DepressionFinderAndRouter	
drainage_area	ChiFinder FastscapeEroder SedDepEroder SteepnessFinder StreamPowerEroder	FlowRouter
flow__link_to_receiver_node	ChiFinder FastscapeEroder SedDepEroder SteepnessFinder StreamPowerEroder	FlowRouter
flow__receiver_node	ChiFinder FastscapeEroder SedDepEroder SteepnessFinder StreamPowerEroder	FlowRouter
flow__sink_flag		FlowRouter
flow__upstream_node_order	ChiFinder FastscapeEroder SedDepEroder SteepnessFinder StreamPowerEroder	FlowRouter
lithosphere__overlying_pressure_increment	Flexure	
lithosphere_surface__elevation_increment		Flexure gFlex
plant__age		VegCA
plant__live_index		VegCA
radiation__incoming_shortwave_flux		PotentialEvapotranspiration Radiation
radiation__net_flux		PotentialEvapotranspiration
radiation__net_longwave_flux		PotentialEvapotranspiration
radiation__net_shortwave_flux		PotentialEvapotranspiration Radiation
radiation__ratio_to_flat_surface	PotentialEvapotranspiration	Radiation
rainfall__daily_depth	SoilMoisture	
sediment_fill__depth		SinkFiller
soil_moisture__initial_saturation_fraction	SoilMoisture	
soil_moisture__root_zone_leakage		SoilMoisture
soil_moisture__saturation_fraction		SoilMoisture
soil_water_infiltration__depth	SoilInfiltrationGreenAmpt	SoilInfiltrationGreenAmpt
surface__evapotranspiration	Vegetation	SoilMoisture
surface__potential_evapotranspiration_30day_mean	Vegetation	
surface__potential_evapotranspiration_rate	SoilMoisture Vegetation	PotentialEvapotranspiration
surface__runoff		SoilMoisture

(continues)

(continued)

Field name	Used by	Provided by
surface_load__stress	gFlex	
surface_water__depth	KinematicWaveRengers OverlandFlow OverlandFlowBates SoilInfiltrationGreenAmpt	KinematicWaveRengers OverlandFlow OverlandFlowBates SoilInfiltrationGreenAmpt
surface_water__discharge	DetachmentLtdErosion	FlowRouter KinematicWaveRengers OverlandFlow OverlandFlowBates
surface_water__velocity		KinematicWaveRengers
topographic__elevation	ChiFinder DepressionFinderAndRouter DetachmentLtdErosion FastscapeEroder FlowRouter KinematicWaveRengers LinearDiffuser OverlandFlow OverlandFlowBates PerronNLDiffuse Radiation SedDepEroder SinkFiller SteepnessFinder StreamPowerEroder	DetachmentLtdErosion FastscapeEroder gFlex LinearDiffuser PerronNLDiffuse SedDepEroder SinkFiller StreamPowerEroder
topographic__gradient		LinearDiffuser
topographic__slope	DetachmentLtdErosion	
topographic__steepest_slope	ChiFinder SedDepEroder SteepnessFinder StreamPowerEroder	FlowRouter
hillslope_sediment__unit_volume_flux		LinearDiffuser
vegetation__cover_fraction	SoilMoisture	
vegetation__cumulative_water_stress	VegCA	Vegetation
vegetation__dead_biomass		Vegetation
vegetation__dead_leaf_area_index		Vegetation
vegetation__live_biomass		Vegetation
vegetation__live_leaf_area_index	SoilMoisture	Vegetation
vegetation__plant_functional_type	SoilMoisture VegCA Vegetation	
vegetation__water_stress	Vegetation	SoilMoisture
water__unit_flux_in	FlowRouter	
water_surface__gradient		OverlandFlow OverlandFlowBates

Script S2. Code to examine run times of simple stream power models on two Landlab grid types (Fig. 11).

```
import numpy as np
from matplotlib.pyplot import figure, show, plot, xlabel, ylabel, ylim
from landlab import RasterModelGrid, HexModelGrid
from landlab.components import StreamPowerEroder, FlowRouter, \
    PrecipitationDistribution
from landlab import imshow_grid
from time import time

uplift_rate = 0.001
side_list = [5, 10, 15, 20, 30, 40, 50, 75, 100, 150]
total_time_listr = []
total_time_listh = []
loop_time_listr = []
loop_time_listh = []
num_repeats = 5

for side in side_list:
    print(side)
    for i in range(2):
        temp_tottime = []
        temp_looptime = []
        for j in range(num_repeats):
            time_0 = time()
            if i == 0:
                mg = RasterModelGrid((side, side), 100.)
            else:
                mg = HexModelGrid(side, side, 100., shape='rect')

            mg_noise = np.random.rand(mg.number_of_nodes)/1000.

            zr = mg.add_zeros('node', 'topographic__elevation')
            zr += mg_noise
            Qr = mg.add_empty('node', 'surface_water__discharge')

            runoff_rater = mg.add_ones('node', 'water__unit_flux_in')

            frr = FlowRouter(mg)
            # ^water__unit_flux_in gets automatically ingested
            spr = StreamPowerEroder(mg, K_sp=1.e-5, threshold_sp=1.e-6,
                                    use_Q='surface_water__discharge')
            precip = PrecipitationDistribution(
                mean_storm_depth=5000., mean_storm_duration=100.,
                mean_interstorm_duration=900., total_t=3.e6)

            time_in = time()
            for (dt, runoff) in \
                    precip.yield_storm_interstorm_duration_intensity():
                zr[mg.core_nodes] += uplift_rate*dt
                if runoff > 0.:
                    runoff_rater.fill(runoff)
                    frr.run_one_step()
                    spr.run_one_step(dt)
```

```
                    # raincount += 1
                    # print(raincount, dt, runoff)
            time_out = time()
            temp_tottime.append(time_out-time_0)
            temp_looptime.append(time_out-time_in)
        tottime = np.mean(temp_tottime)
        looptime = np.mean(temp_looptime)
        if i == 0:
            loop_time_listr.append(looptime)
            total_time_listr.append(tottime)
        else:
            loop_time_listh.append(looptime)
            total_time_listh.append(tottime)

np.savetxt('side_list', np.array(side_list))
np.savetxt('loop_time_listr', np.array(loop_time_listr))
np.savetxt('total_time_listr', np.array(total_time_listr))
np.savetxt('loop_time_listh', np.array(loop_time_listh))
np.savetxt('total_time_listh', np.array(total_time_listh))

side_list = np.loadtxt('side_list')
loop_time_listr = np.loadtxt('loop_time_listr')
total_time_listr = np.loadtxt('total_time_listr')
loop_time_listh = np.loadtxt('loop_time_listh')
total_time_listh = np.loadtxt('total_time_listh')
s = 12.
figure('runtimes')
plot(np.array(side_list)**2, loop_time_listr, 'b+', markersize=s)
plot(np.array(side_list)**2, total_time_listr, 'D',
     markerfacecolor='none', markeredgecolor='b', markersize=s)
plot(np.array(side_list)**2, loop_time_listh, 'rx', markersize=s)
plot(np.array(side_list)**2, total_time_listh, 's',
     markerfacecolor='none', markeredgecolor='r', markersize=s)
xlabel('Number of nodes')
ylabel('Time (s)')

figure('overhead')
overhead_r = np.array(total_time_listr) - np.array(loop_time_listr)
overhead_h = np.array(total_time_listh) - np.array(loop_time_listh)
plot(np.array(side_list)**2, overhead_r, 'bo', markersize=s)
plot(np.array(side_list)**2, overhead_h, 'r^', markersize=s)
ylim([0, 2.1])
xlabel('Number of nodes')
ylabel('Time (s)')

show()
```

Script S3. Code to run a simple stream power model in Landlab, incorporating both storms and a threshold, on two different grid types (Fig. 12).

```
import numpy as np
from matplotlib.pyplot import figure, show, loglog, xlim, ylim, xlabel, ylabel
from landlab import RasterModelGrid, HexModelGrid
from landlab.components import StreamPowerEroder, FlowRouter, \
    PrecipitationDistribution
from landlab import imshow_grid
from copy import deepcopy

# This script is to make fig 11
side = 100
uplift_rate = 0.001
gridlist = []

for i in range(2):
    if i == 0:
        mg = RasterModelGrid((side, side), 100.)
    else:
        mg = HexModelGrid(side, side, 100., shape='rect')

    # add initial noise to produce convergent flow from the initial conditions
    np.random.seed(0)  # so our figures are reproducible
    mg_noise = np.random.rand(mg.number_of_nodes)/1000.

    # set up the input fields
    zr = mg.add_zeros('node', 'topographic__elevation')
    zr += mg_noise
    Qr = mg.add_empty('node', 'surface_water__discharge')
    runoff_rater = mg.add_ones('node', 'water__unit_flux_in')

    # Landlab sets fixed elevation boundary conditions by default. This is
    # what we want, so we will not modify these here.

    # instantiate the components:
    frr = FlowRouter(mg)  # water__unit_flux_in gets automatically ingested
    spr = StreamPowerEroder(
        mg, K_sp=1.e-5, m_sp=0.5, n_sp=1., threshold_sp=1.e-6,
        use_Q='surface_water__discharge')
    # the `use_Q` flag tells the StreamPowerEroder to use discharge defined in
    # the field 'surface_water__discharge' as the first term in the stream
    # power law, not the drainage area, as is sometimes also seen.
    precip = PrecipitationDistribution(
        mean_storm_depth=5000., mean_storm_duration=100.,
        mean_interstorm_duration=900., total_t=3.e6)

    raincount = 0  # this flag lets us see how many rain events have occured
    for (dt, runoff) in precip.yield_storm_interstorm_duration_intensity():
        zr[mg.core_nodes] += uplift_rate*dt
        if runoff > 0.:
            runoff_rater.fill(runoff)
            frr.run_one_step()
            spr.run_one_step(dt)
            raincount += 1
```

```
            print(raincount, dt, runoff)
        # this loop will terminate automatically, thanks to the generator
        # method we're calling from the `precip` class object.

    # Do some plotting. First the topography:
    figure('topo ' + str(i))
    imshow_grid(mg, zr, grid_units=('m', 'm'), var_name='Elevation (m)')

    # then some slope-area plots, for checking:
    figure('S-A_all')
    loglog(mg.at_node['drainage_area'],
           mg.at_node['topographic__steepest_slope'], 'x')
    figure('S-A_interior_only ' + str(i))
    edge = np.unique(mg.neighbors_at_node[mg.boundary_nodes, :])
    not_edge = np.in1d(mg.nodes.flatten(), edge, assume_unique=True,
                       invert=True)
    loglog(mg.at_node['drainage_area'][not_edge],
           mg.at_node['topographic__steepest_slope'][not_edge], 'x')
    xlim([1.e3, 1.e8])
    xlabel('Topographic slope')
    ylabel('Drainage area (m^2)')

    # save the data to a list so we can inspect both grid types at our leisure
    # if this script is run from an interactive Python environment like
    # iPython:
    gridlist.append(deepcopy(mg))

# show all the plots we have:
show()

# just to produce the figures:
i = 0
for mg in gridlist:
    figure('topo ' + str(i))
    imshow_grid(mg, 'topographic__elevation', grid_units=('m', 'm'),
                var_name='Elevation (m)')
    figure('S-A ' + str(i))
    edge = np.unique(mg.neighbors_at_node[mg.boundary_nodes, :])
    not_edge = np.in1d(mg.nodes.flatten(), edge, assume_unique=True,
                       invert=True)
    loglog(mg.at_node['drainage_area'][not_edge],
           mg.at_node['topographic__steepest_slope'][not_edge], 'x')
    xlim([1.e3, 1.e7])
    ylabel('Topographic slope')
    xlabel('Drainage area (m^2)')
    figure('S-A')
    loglog(mg.at_node['drainage_area'][not_edge],
           mg.at_node['topographic__steepest_slope'][not_edge], 'x')
    xlim([1.e3, 1.e7])
    ylabel('Topographic slope')
    xlabel('Drainage area (m^2)')
    i += 1
show()
```

Script S4. Code to run a coupled stream power-hillslope diffusion model in Landlab on two different grid types (Fig. 12).

```
import numpy as np
from matplotlib.pyplot import figure, show, loglog, xlim, ylim, xlabel, ylabel
from landlab import RasterModelGrid, HexModelGrid
from landlab.components import StreamPowerEroder, FlowRouter, \
    PrecipitationDistribution, LinearDiffuser, DepressionFinderAndRouter
from landlab import imshow_grid
from copy import deepcopy

side = 100
uplift_rate = 0.001
gridlist = []

for i in range(2):
    if i == 0:
        mg = RasterModelGrid((side, side), 100.)
    else:
        mg = HexModelGrid(side, side, 100., shape='rect')

    # add initial noise to produce convergent flow from the initial conditions
    np.random.seed(0)  # so our figures are reproducible
    mg_noise = np.random.rand(mg.number_of_nodes)/1000.

    # set up the input fields
    zr = mg.add_zeros('node', 'topographic__elevation')
    zr += mg_noise
    Qr = mg.add_empty('node', 'surface_water__discharge')
    runoff_rater = mg.add_ones('node', 'water__unit_flux_in')

    # Landlab sets fixed elevation boundary conditions by default. This is
    # what we want, so we will not modify these here.

    # instantiate the components:
    frr = FlowRouter(mg)  # water__unit_flux_in gets automatically ingested
    spr = StreamPowerEroder(
        mg, K_sp=1.e-5, m_sp=0.5, n_sp=1., threshold_sp=1.e-6,
        use_Q='surface_water__discharge')
    # the `use_Q` flag tells the StreamPowerEroder to use discharge defined in
    # the field 'surface_water__discharge' as the first term in the stream
    # power law, not the drainage area, as is sometimes also seen.
    dfn = LinearDiffuser(mg, linear_diffusivity=0.05)
    lake = DepressionFinderAndRouter(mg)
    precip = PrecipitationDistribution(
        mean_storm_depth=5000., mean_storm_duration=100.,
        mean_interstorm_duration=900., total_t=3.e6)

    raincount = 0  # this flag lets us see how many rain events have occured
    for (dt, runoff) in precip.yield_storm_interstorm_duration_intensity():
        zr[mg.core_nodes] += uplift_rate*dt
        dfn.run_one_step(dt)  # hillslopes always diffusive, even when dry
        if runoff > 0.:
            runoff_rater.fill(runoff)
            frr.run_one_step()
```

```
            lake.map_depressions()
            spr.run_one_step(dt, flooded_nodes=lake.lake_at_node)
            raincount += 1
            print(raincount, dt, runoff)
        # this loop will terminate automatically, thanks to the generator
        # method we're calling from the `precip` class object.

    # Do some plotting. First the topography:
    figure('topo ' + str(i))
    imshow_grid(mg, zr, grid_units=('m', 'm'), var_name='Elevation (m)')

    # then some slope-area plots, for checking:
    figure('S-A_all')
    loglog(mg.at_node['drainage_area'],
           mg.at_node['topographic__steepest_slope'], 'x')
    figure('S-A_interior_only ' + str(i))
    edge = np.unique(mg.neighbors_at_node[mg.boundary_nodes, :])
    not_edge = np.in1d(mg.nodes.flatten(), edge, assume_unique=True,
                       invert=True)
    loglog(mg.at_node['drainage_area'][not_edge],
           mg.at_node['topographic__steepest_slope'][not_edge], 'x')
    xlim([1.e3, 1.e7])
    xlabel('Topographic slope')
    ylabel('Drainage area (m^2)')

    # save the data to a list so we can inspect both grid types at our leisure
    # if this script is run from an interactive Python environment like
    # iPython:
    gridlist.append(deepcopy(mg))

# show all the plots we have:
show()

# just to produce the figures:
i = 0
for mg in gridlist:
    figure('topo ' + str(i))
    imshow_grid(mg, 'topographic__elevation', grid_units=('m', 'm'),
                var_name='Elevation (m)')
    figure('S-A ' + str(i))
    edge = np.unique(mg.neighbors_at_node[mg.boundary_nodes, :])
    not_edge = np.in1d(mg.nodes.flatten(), edge, assume_unique=True,
                       invert=True)
    loglog(mg.at_node['drainage_area'][not_edge],
           mg.at_node['topographic__steepest_slope'][not_edge], 'x')
    xlim([1.e3, 1.e7])
    ylabel('Topographic slope')
    xlabel('Drainage area (m^2)')
    figure('S-A')
    loglog(mg.at_node['drainage_area'][not_edge],
           mg.at_node['topographic__steepest_slope'][not_edge], 'x')
    xlim([1.e3, 1.e7])
    ylabel('Topographic slope')
    xlabel('Drainage area (m^2)')
    i += 1
show()
```

Script S5. Code to run an ecohydrology model in Landlab.

```
# Authors: Sai Nudurupati & Erkan Istanbulluoglu, 21May15
# Edited: 15Jul16 - to conform to Landlab version 1.
# A companion interactive tutorial an be found at: landlab/tutorials/
# ...ecohydrology/cellular_automaton_vegetation_flat_surface.ipynb

import os
import time
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from landlab import load_params, RasterModelGrid
from landlab.plot import imshow_grid
from landlab.components import (PrecipitationDistribution, Radiation,
                                PotentialEvapotranspiration, SoilMoisture,
                                Vegetation, VegCA)

GRASS = 0
SHRUB = 1
TREE = 2
BARE = 3
SHRUBSEEDLING = 4
TREESEEDLING = 5


def compose_veg_grid(grid, percent_bare=0.4, percent_grass=0.2,
                     percent_shrub=0.2, percent_tree=0.2):
    """Compose spatially distribute PFT."""
    no_cells = grid.number_of_cells
    shrub_point = int(percent_bare * no_cells)
    tree_point = int((percent_bare + percent_shrub) * no_cells)
    grass_point = int((1 - percent_grass) * no_cells)

    veg_grid = np.full(grid.number_of_cells, BARE, dtype=int)
    veg_grid[shrub_point:tree_point] = SHRUB
    veg_grid[tree_point:grass_point] = TREE
    veg_grid[grass_point:] = GRASS

    np.random.shuffle(veg_grid)
    return veg_grid


def initialize(data, grid, grid1):
    """Initialize random plant type field.

    Plant types are defined as the following:

    *  GRASS = 0
    *  SHRUB = 1
    *  TREE = 2
    *  BARE = 3
    *  SHRUBSEEDLING = 4
    *  TREESEEDLING = 5
    """
```

```
    grid1.at_cell['vegetation__plant_functional_type'] = compose_veg_grid(
        grid1, percent_bare=data['percent_bare_initial'],
        percent_grass=data['percent_grass_initial'],
        percent_shrub=data['percent_shrub_initial'],
        percent_tree=data['percent_tree_initial'])

    # Assign plant type for representative ecohydrologic simulations
    grid.at_cell['vegetation__plant_functional_type'] = np.arange(6)
    grid1.at_node['topographic__elevation'] = np.full(grid1.number_of_nodes,
                                                       1700.)
    grid.at_node['topographic__elevation'] = np.full(grid.number_of_nodes,
                                                      1700.)
    precip_dry = PrecipitationDistribution(
        mean_storm_duration=data['mean_storm_dry'],
        mean_interstorm_duration=data['mean_interstorm_dry'],
        mean_storm_depth=data['mean_storm_depth_dry'])
    precip_wet = PrecipitationDistribution(
        mean_storm_duration=data['mean_storm_wet'],
        mean_interstorm_duration=data['mean_interstorm_wet'],
        mean_storm_depth=data['mean_storm_depth_wet'])

    radiation = Radiation(grid)
    pet_tree = PotentialEvapotranspiration(grid, method=data['PET_method'],
                                           MeanTmaxF=data['MeanTmaxF_tree'],
                                           delta_d=data['DeltaD'])
    pet_shrub = PotentialEvapotranspiration(grid, method=data['PET_method'],
                                            MeanTmaxF=data['MeanTmaxF_shrub'],
                                            delta_d=data['DeltaD'])
    pet_grass = PotentialEvapotranspiration(grid, method=data['PET_method'],
                                            MeanTmaxF=data['MeanTmaxF_grass'],
                                            delta_d=data['DeltaD'])
    soil_moisture = SoilMoisture(grid, **data) # Soil Moisture object
    vegetation = Vegetation(grid, **data) # Vegetation object
    vegca = VegCA(grid1, **data) # Cellular automaton object

    # Initializing inputs for Soil Moisture object
    grid.at_cell['vegetation__live_leaf_area_index'] = (
        1.6 * np.ones(grid.number_of_cells))
    grid.at_cell['soil_moisture__initial_saturation_fraction'] = (
        0.59 * np.ones(grid.number_of_cells))

    return (precip_dry, precip_wet, radiation, pet_tree, pet_shrub,
            pet_grass, soil_moisture, vegetation, vegca)


def empty_arrays(n, grid, grid1):
    precip = np.empty(n) # Record precipitation
    inter_storm_dt = np.empty(n) # Record inter storm duration
    storm_dt = np.empty(n) # Record storm duration
    time_elapsed = np.empty(n) # To record time elapsed from start of simulation

    # Cumulative Water Stress
    veg_type = np.empty([n / 55, grid1.number_of_cells], dtype=int)
    daily_pet = np.zeros([365, grid.number_of_cells])
    rad_factor = np.empty([365, grid.number_of_cells])
    EP30 = np.empty([365, grid.number_of_cells])
```

```

    # 30 day average PET to determine season
    pet_threshold = 0  # Initializing pet_threshold to ETThresholddown
    return (precip, inter_storm_dt, storm_dt, time_elapsed, veg_type,
            daily_pet, rad_factor, EP30, pet_threshold)


def create_pet_lookup(radiation, pet_tree, pet_shrub, pet_grass, daily_pet,
                      rad_factor, EP30, grid):
    for i in range(0, 365):
        pet_tree.update(float(i) / 365.25)
        pet_shrub.update(float(i) / 365.25)
        pet_grass.update(float(i) / 365.25)
        daily_pet[i] = [pet_grass._PET_value, pet_shrub._PET_value,
                   pet_tree._PET_value, 0., pet_shrub._PET_value,
                   pet_tree._PET_value]
        radiation.update(float(i) / 365.25)
        rad_factor[i] = grid.at_cell['radiation__ratio_to_flat_surface']

        if i < 30:
            if i == 0:
                EP30[0] = daily_pet[0]
            else:
                EP30[i] = np.mean(daily_pet[:i], axis=0)
        else:
            EP30[i] = np.mean(daily_pet[i - 30:i], axis=0)


def save(sim, inter_storm_dt, storm_dt, precip, veg_type, yrs,
         walltime, time_elapsed):
    np.save(sim + '_Tb', inter_storm_dt)
    np.save(sim + '_Tr', storm_dt)
    np.save(sim + '_P', precip)
    np.save(sim + '_VegType', veg_type)
    np.save(sim + '_Years', yrs)
    np.save(sim + '_Time_Consumed_minutes', walltime)
    np.save(sim + '_CurrentTime', time_elapsed)


def plot(sim, grid, veg_type, yrs, yr_step=10):
    pic = 0
    years = range(0, yrs)
    cmap = mpl.colors.ListedColormap(
        ['green', 'red', 'black', 'white', 'red', 'black'])
    bounds = [-0.5, 0.5, 1.5, 2.5, 3.5, 4.5, 5.5]
    norm = mpl.colors.BoundaryNorm(bounds, cmap.N)
    print 'Plotting cellular field of Plant Functional Type'
    print 'Green - Grass; Red - Shrubs; Black - Trees; White - Bare'

    # Plot images to make gif.
    for year in range(0, yrs, yr_step):
        filename = 'year_' + "%05d" % year
        pic += 1
        plt.figure(pic, figsize=(10, 8))
        imshow_grid(grid, veg_type[year], values_at='cell', cmap=cmap,
                    grid_units=('m', 'm'), norm=norm, limits=[0, 5],
```

```
                        allow_colorbar=False)
            plt.title(filename, weight='bold', fontsize=22)
            plt.xlabel('X (m)', weight='bold', fontsize=18)
            plt.ylabel('Y (m)', weight='bold', fontsize=18)
            plt.xticks(fontsize=14, weight='bold')
            plt.yticks(fontsize=14, weight='bold')
            plt.savefig(sim + '_' + filename)

        grass_cov = np.empty(yrs)
        shrub_cov = np.empty(yrs)
        tree_cov = np.empty(yrs)
        grid_size = float(veg_type.shape[1])

        for x in range(0, yrs):
            grass_cov[x] = (veg_type[x][veg_type[x] == GRASS].size /
                            grid_size) * 100
            shrub_cov[x] = ((veg_type[x][veg_type[x] == SHRUB].size / grid_size) *
                            100 + (veg_type[x][veg_type[x] == SHRUBSEEDLING].size /
                            grid_size) * 100)
            tree_cov[x] = ((veg_type[x][veg_type[x] == TREE].size / grid_size) *
                           100 + (veg_type[x][veg_type[x] == TREESEEDLING].size /
                           grid_size) * 100)

        pic += 1
        plt.figure(pic, figsize=(10, 8))
        plt.plot(years, grass_cov, '-g', label='Grass', linewidth=4)
        plt.hold(True)
        plt.plot(years, shrub_cov, '-r', label='Shrub', linewidth=4)
        plt.hold(True)
        plt.plot(years, tree_cov, '-k', label='Tree', linewidth=4)
        plt.ylabel('% Area Covered by Plant Type', weight='bold', fontsize=18)
        plt.xlabel('Time in years', weight='bold', fontsize=18)
        plt.xticks(fontsize=12, weight='bold')
        plt.yticks(fontsize=12, weight='bold')
        plt.legend(loc=0, prop={'size': 16, 'weight': 'bold'})
        plt.savefig(sim + '_percent_cover')


# Now a script to drive the model:

grid1 = RasterModelGrid((100, 100), spacing=(5., 5.))
grid = RasterModelGrid((5, 4), spacing=(5., 5.))

# Create dictionary that holds the inputs
data = load_params('inputs_vegetation_ca.yaml')

(precip_dry, precip_wet, radiation, pet_tree, pet_shrub,
 pet_grass, soil_moisture, vegetation, vegca) = initialize(data, grid, grid1)

n_years = 2000 # Approx number of years for model to run

# Calculate approximate number of storms per year
fraction_wet = (data['doy__end_of_monsoon'] -
                data['doy__start_of_monsoon']) / 365.
fraction_dry = 1 - fraction_wet
no_of_storms_wet = 8760 * fraction_wet / (data['mean_interstorm_wet'] +
```

```
                                          data['mean_storm_wet'])
no_of_storms_dry = 8760 * fraction_dry / (data['mean_interstorm_dry'] +
                                          data['mean_storm_dry'])
n = int(n_years * (no_of_storms_wet + no_of_storms_dry))

(precip, inter_storm_dt, storm_dt, time_elapsed, veg_type, daily_pet,
 rad_factor, EP30, pet_threshold) = empty_arrays(n, grid, grid1)

create_pet_lookup(radiation, pet_tree, pet_shrub, pet_grass,  daily_pet,
                  rad_factor, EP30, grid)

# Represent current time in years
current_time = 0 # Start from first day of Jan

# Keep track of run time for simulation - optional
wallclock_start = time.clock() # Recording time taken for simulation

# declaring few variables that will be used in the storm loop
time_check = 0. # Buffer to store current_time at previous storm
yrs = 0 # Keep track of number of years passed
water_stress = 0. # Buffer for Water Stress
Tg = 270 # Growing season in days

# Run storm Loop
for i in range(n):
    # Update objects

    # Calculate Day of Year (DOY)
    julian = np.int(np.floor((current_time - np.floor(current_time)) * 365.))

    # Generate seasonal storms
    # Wet Season - Jul to Sep - NA Monsoon
    if data['doy__start_of_monsoon'] <= julian <= data['doy__end_of_monsoon']:
        precip_wet.update()
        precip[i] = precip_wet.storm_depth
        storm_dt[i] = precip_wet.storm_duration
        inter_storm_dt[i] = precip_wet.interstorm_duration
    else: # for Dry season
        precip_dry.update()
        precip[i] = precip_dry.storm_depth
        storm_dt[i] = precip_dry.storm_duration
        inter_storm_dt[i] = precip_dry.interstorm_duration

    # Spatially distribute PET and its 30-day-mean (analogous to degree day)
    grid.at_cell[
        'surface__potential_evapotranspiration_rate'] = daily_pet[julian]
    grid.at_cell[
        'surface__potential_evapotranspiration_30day_mean'] = EP30[julian]

    # Assign spatial rainfall data
    grid.at_cell[
        'rainfall__daily_depth'] = np.full(grid.number_of_cells, precip[i])

    # Update soil moisture component
    current_time = soil_moisture.update(current_time, Tr=storm_dt[i],
                                        Tb=inter_storm_dt[i])
```

```

    # Decide whether its growing season or not
    if julian != 364:
        if EP30[julian + 1, 0] > EP30[julian, 0]:
            pet_threshold = 1
            # 1 corresponds to ETThresholdup (begin growing season)
        else:
            pet_threshold = 0
            # 0 corresponds to ETThresholddown (end growing season)

    # Update vegetation component
    vegetation.update(PETThreshold_switch=pet_threshold, Tb=inter_storm_dt[i],
                      Tr=storm_dt[i])

    # Update yearly cumulative water stress data
    water_stress += (grid.at_cell['vegetation__water_stress'] *
                     inter_storm_dt[i] / 24.)

    # Record time (optional)
    time_elapsed[i] = current_time

    # Update spatial PFTs with Cellular Automata rules
    if (current_time - time_check) >= 1.:
        if yrs % 100 == 0:
            print 'Elapsed time = ', yrs, ' years'
        veg_type[yrs] = grid1.at_cell['vegetation__plant_functional_type']
        WS_ = np.choose(veg_type[yrs], water_stress)
        grid1.at_cell['vegetation__cumulative_water_stress'] = WS_ / Tg
        vegca.update()
        time_check = current_time
        water_stress = 0
        yrs += 1

veg_type[yrs] = grid1.at_cell['vegetation__plant_functional_type']

wallclock_stop = time.clock()
walltime = (wallclock_stop - wallclock_start) / 60. # in minutes
print 'Time_consumed = ', walltime, ' minutes'

# Saving
try:
    os.mkdir('output')
except OSError:
    pass
finally:
    os.chdir('output')

save('veg', inter_storm_dt, storm_dt, precip, veg_type, yrs,
     walltime, time_elapsed)

plot('veg', grid1, veg_type, yrs, yr_step=100)
```

This code makes use of the following input text file, named "inputs_vegetation_ca.yaml":


```
### All inputs for Vegetation Cellular Automaton Model built on The Landlab
### can be given here.
### 14Feb2015 - Sai Nudurupati & Erkan Istanbulluoglu
### 15Jul2016 - Updated to comply with Landlab Version 1 naming conventions.

### Vegetation Cellular Automaton Model Input File:

n_short: 6600 # Number of storms for short simulation that plots hydrologic
parameters
n_long_DEM: 1320 # Number of storms for long simulation that operates on single
grid for sloped surface
n_long_flat: 660000 # Number of storms for long simulation that operates on two
grids - flat surface

## Initial Plant Functional Types (PFT) distribution
percent_bare_initial: 0.7 # Initial percentage of cells occupied by bare soil
percent_grass_initial: 0.1 # Initial percentage of cells occupied by grass
percent_shrub_initial: 0.1 # Initial percentage of cells occupied by shrubs
percent_tree_initial: 0.1  # Initial percentage of cells occupied by trees

## Precipitation:

# Dry Season
mean_storm_dry: 2.016 # Mean storm duration (hours)
mean_interstorm_dry: 159.36 # Mean interstorm duration (hours)
mean_storm_depth_dry: 3.07 # Mean storm depth (mm)

# Wet Season
mean_storm_wet: 1.896 # Mean storm duration (hours)
mean_interstorm_wet: 84.24 # Mean interstorm duration (hours)
mean_storm_depth_wet: 4.79 # Mean storm depth (mm)
doy__start_of_monsoon: 182   # Day of the year when the monsoon starts
doy__end_of_monsoon: 273 # Day of the year when the monsoon ends

## PotentialEvapotranspiration:
# Cosine Method
PET_method: Cosine
LT: 0 # Lag between peak TmaxF estimated by cosine method and solar forcing
(days)
DeltaD: 7. # Calibrated difference between
ND: 365. # Number of days in the year (days)
MeanTmaxF_grass: 5.15 # Mean annual rate of TmaxF (mm/d)
MeanTmaxF_shrub: 3.77 # Mean annual rate of TmaxF (mm/d)
MeanTmaxF_tree: 4.96 # Mean annual rate of TmaxF (mm/d)

# TmaxF - Estimated maximum evapotranspiration as a function of DOY
# using Penman Monteith method for historical weather

## Soil Moisture:

runon: 0. # Runon from higher elevations (mm)
f_bare: 0.7 # Fraction to partition PET for bare soil (None)
```

```
# Grass

VEGTYPE_grass: 0 # Integer value to infer Vegetation Type
intercept_cap_grass: 1. # Full canopy interception capacity  (mm)
zr_grass: 0.3 # Root depth (m)
I_B_grass: 20. # Infiltration capacity of bare soil (mm/h)
I_V_grass: 24. # Infiltration capacity of vegetated soil (mm/h)
pc_grass: 0.43 # Soil porosity (None)
fc_grass: 0.56 # Soil saturation degree at field capacity (None)
sc_grass: 0.33 # Soil saturation degree at stomatal closure (None)
wp_grass: 0.13 # Soil saturation degree at wilting point (None)
hgw_grass: 0.1 # Soil saturation degree at hygroscopic point (None)
beta_grass: 13.8 # Deep percolation constant = 2*b+4  where b is water retention
parameter

# Shrub

VEGTYPE_shrub: 1 # Integer value to infer Vegetation Type
intercept_cap_shrub: 1.5 # Full canopy interception capacity  (mm)
zr_shrub: 0.5 # Root depth (m)
I_B_shrub: 20. # Infiltration capacity of bare soil (mm/h)
I_V_shrub: 40. # Infiltration capacity of vegetated soil (mm/h)
pc_shrub: 0.43 # Soil porosity (None)
fc_shrub: 0.56 # Soil saturation degree at field capacity (None)
sc_shrub: 0.24 # Soil saturation degree at stomatal closure (None)
wp_shrub: 0.13 # Soil saturation degree at wilting point (None)
hgw_shrub: 0.1 # Soil saturation degree at hygroscopic point (None)
beta_shrub: 13.8 # Deep percolation constant = 2*b+4  where b is water retention
parameter

# Tree

VEGTYPE_tree: 2 # Integer value to infer Vegetation Type
intercept_cap_tree: 2. # Full canopy interception capacity  (mm)
zr_tree: 1.3 # Root depth (m)
I_B_tree: 20. # Infiltration capacity of bare soil (mm/h)
I_V_tree: 40. # Infiltration capacity of vegetated soil (mm/h)
pc_tree: 0.43 # Soil porosity (None)
fc_tree: 0.56 # Soil saturation degree at field capacity (None)
sc_tree: 0.22 # Soil saturation degree at stomatal closure (None)
wp_tree: 0.15 # Soil saturation degree at wilting point (None)
hgw_tree: 0.1 # Soil saturation degree at hygroscopic point (None)
beta_tree: 13.8 # Deep percolation constant = 2*b+4  where b is water retention
parameter

# Bare Soil

VEGTYPE_bare: 3 # Integer value to infer Vegetation Type
intercept_cap_bare: 1. # Full canopy interception capacity  (mm)
zr_bare: 0.15 # Root depth (m)
I_B_bare: 20. # Infiltration capacity of bare soil (mm/h)
I_V_bare: 20. # Infiltration capacity of vegetated soil (mm/h)
pc_bare: 0.43 # Soil porosity (None)
fc_bare: 0.56 # Soil saturation degree at field capacity (None)
sc_bare: 0.33 # Soil saturation degree at stomatal closure (None)
```

```
wp_bare: 0.13 # Soil saturation degree at wilting point (None)
hgw_bare: 0.1 # Soil saturation degree at hygroscopic point (None)
beta_bare: 13.8 # Deep percolation constant


## Vegetation Dynamics:

Blive_init: 102.
Bdead_init: 450.
PET_growth_threshold: 3.8 # PET threshold for growing season (mm/d)
PET_dormancy_threshold: 6.8  # PET threshold for dormant season (mm/d)
Tdmax: 10. # Constant for dead biomass loss adjustment (mm/d)
w: 0.55 # Conversion factor of CO2 to dry biomass (Kg DM/Kg CO2)

# Grass

WUE_grass: 0.01 # Water use efficiency KgCO2Kg-1H2O
cb_grass: 0.0047 # Specific leaf area for green/live biomass (m2 leaf g-1 DM)
cd_grass: 0.009 # Specific leaf area for dead biomass (m2 leaf g-1 DM)
ksg_grass: 0.012 # Senescence coefficient of green/live biomass (d-1)
kdd_grass: 0.013 # Decay coefficient of aboveground dead biomass (d-1)
kws_grass: 0.02 # Maximum drought induced foliage loss rate (d-1)
LAI_max_grass: 2. # Maximum leaf area index (m2/m2)
LAIR_max_grass: 2.88 # Reference leaf area index (m2/m2)

# Shrub

WUE_shrub: 0.0025 # Water use efficiency KgCO2Kg-1H2O
cb_shrub: 0.004 # Specific leaf area for green/live biomass (m2 leaf g-1 DM)
cd_shrub: 0.01 # Specific leaf area for dead biomass (m2 leaf g-1 DM)
ksg_shrub: 0.002 # Senescence coefficient of green/live biomass (d-1)
kdd_shrub: 0.013 # Decay coefficient of aboveground dead biomass (d-1)
kws_shrub: 0.02 # Maximum drought induced foliage loss rate (d-1)
LAI_max_shrub: 2. # Maximum leaf area index (m2/m2)
LAIR_max_shrub: 2. # Reference leaf area index (m2/m2)

# Tree

WUE_tree: 0.0045 # Water use efficiency KgCO2Kg-1H2O
cb_tree: 0.004 # Specific leaf area for green/live biomass (m2 leaf g-1 DM)
cd_tree: 0.01 # Specific leaf area for dead biomass (m2 leaf g-1 DM)
ksg_tree: 0.002 # Senescence coefficient of green/live biomass (d-1)
kdd_tree: 0.013 # Decay coefficient of aboveground dead biomass (d-1)
kws_tree: 0.01 # Maximum drought induced foliage loss rate (d-1)
LAI_max_tree: 4. # Maximum leaf area index (m2/m2)
LAIR_max_tree: 4. # Reference leaf area index (m2/m2)

# Bare

WUE_bare: 0.01 # Water use efficiency KgCO2Kg-1H2O
cb_bare: 0.0047 # Specific leaf area for green/live biomass (m2 leaf g-1 DM)
cd_bare: 0.009 # Specific leaf area for dead biomass (m2 leaf g-1 DM)
ksg_bare: 0.012 # Senescence coefficient of green/live biomass (d-1)
kdd_bare: 0.013 # Decay coefficient of aboveground dead biomass (d-1)
kws_bare: 0.02 # Maximum drought induced foliage loss rate (d-1)
LAI_max_bare: 0.01 # Maximum leaf area index (m2/m2)
```

```
LAIR_max_bare: 0.01 # Reference leaf area index (m2/m2)


## Cellular Automaton Vegetation:

# Grass

Pemaxg: 0.35 # Maximal establishment probability
ING: 2 # Parameter to define allelopathic effect on grass from cresotebush
ThetaGrass: 0.62 # Drought resistant threshold
PmbGrass: 0.05               # Background mortality probability

# Shrub

Pemaxsh: 0.2 # Maximal establishment probability
ThetaShrub: 0.78 # Drought resistant threshold
PmbShrub: 0.03 # Background mortality probability
tpmaxShrub: 600 # Maximum age (yr)

# Tree

Pemaxtr: 0.25 # Maximal establishment probability
ThetaTree: 0.72 # Drought resistant threshold
PmbTree: 0.01 # Background mortality probability
tpmaxTree: 350 # Maximum age (yr)

# ShrubSeedling

ThetaShrubSeedling: 0.64 # Drought resistant threshold
PmbShrubSeedling: 0.03 # Background mortality probability
tpmaxShrubSeedling: 18 # Maximum age (yr)

# TreeSeedling

ThetaTreeSeedling: 0.64 # Drought resistant threshold
PmbTreeSeedling: 0.03 # Background mortality probability
tpmaxTreeSeedling: 18 # Maximum age (yr)
```

Script S6. Code to run an surface runoff model in Landlab.

```
from landlab.io import read_esri_ascii
from landlab.components import SinkFiller, OverlandFlow
(mg, z) = read_esri_ascii('Watershed_DEM.asc',
                          name='topographic__elevation')
mg.set_watershed_boundary_condition(z)
sf = SinkFiller(mg, routing='D4', apply_slope=True, fill_slope=1.e-5)
sf.fill_pits()
of = OverlandFlow(mg, steep_slopes=True)
elapsed_time = 0.
storm_duration = 3600.   # in seconds
model_run_time = 84600.  # in seconds
starting_precip = 25.    # in mm/h
while elapsed_time < model_run_time:
    if elapsed_time < storm_duration:
        of.rainfall_intensity = starting_precip * 2.7778e-7  # mm/h to m/s
    else:
        of.rainfall_intensity = 0.
    of.run_one_step()  # this component can select its own timestep
    elapsed_time += of.dt
```