ASTR 3750 Course Project: Impact Crater Saturation Simulation

Christopher Leighton
December 1st, 2016

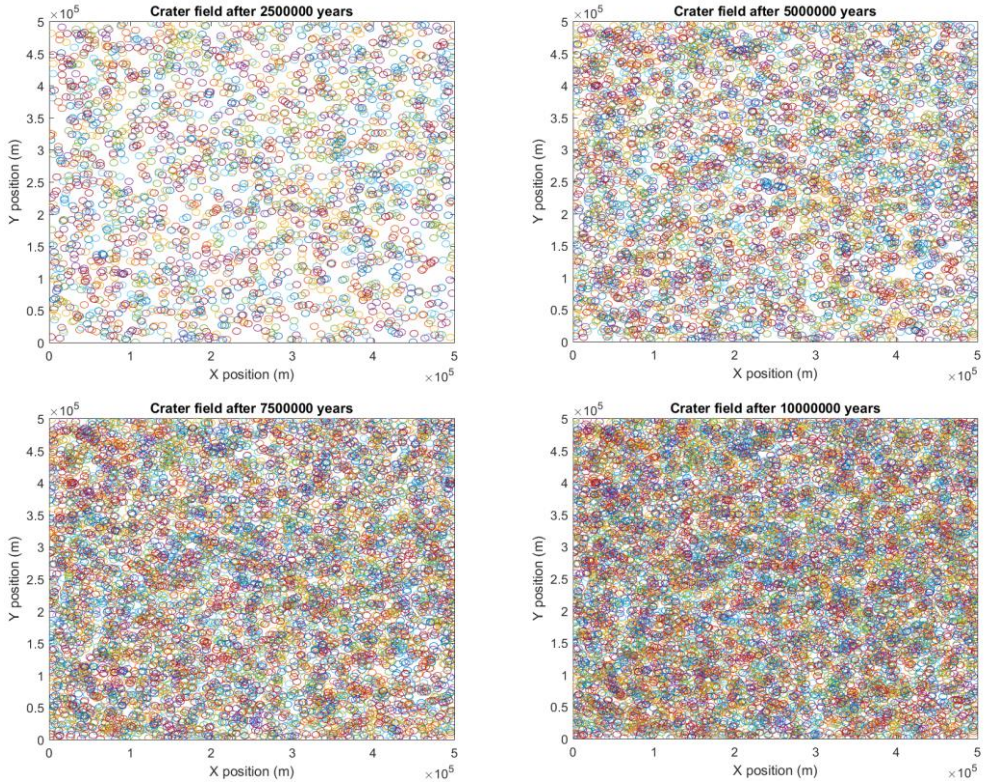
1. Constant Crater Size

a. I made the following assumptions for part 1.
   i. All craters placed on the field will have a diameter of 10 kilometers. This is quite a bad assumption, seeing as there's an enormous variability in actual crater size. This is the primary assumption to be addressed in part 2.
   ii. A crater will be considered as having been erased if more than 50% of its area has been occluded by newer craters. This isn't a great assumption, as many craters that are more than half erased can be recognized. Then again, I'm sure that there are many craters that are less than half erased that are unrecognizable (perhaps due to things like erosion). A line had to be drawn somewhere. It's worth pointing out that in my code, this required percentage can be changed extremely easily.
   iii. A new crater of diameter 10 kilometers will be created every 1000 years. This seems like a relatively safe assumption, though I can't speak to the actual time frequency of 10 kilometer craters.
   iv. Error when calculating overlapping areas by point counting method (see below) is negligible. My code calculates the overlapping area between two craters by a brute force method, but instead of counting every single square meter, it divides the overlapping area into a series of large squares, the widths of which are on the order of hundreds of meters (this refers to the *accu_fact* variable in my code). If my intuition is right, the error associated with this calculation increases when the actual area of intersection is small. Graphically, if we have a tiny actual area and we have several points calculated to be inside that region, the square areas calculated as a result of the number of internal points will likely extend outside of the actual area. Looping over every single point in the overlapping region would be ideal, and I attempted to do so, but it turned out to be far, far too computationally expensive, as my program is written with five nested loops. There's probably a more efficient way to have written it that would make looping over every point feasible, but I don't know how that would work.

v. The craters are all circular; i.e. oddly-shaped craters are ignored. This seems like a completely safe assumption. As we have learned in class, even highly oblique impacts tend to result in circular craters.

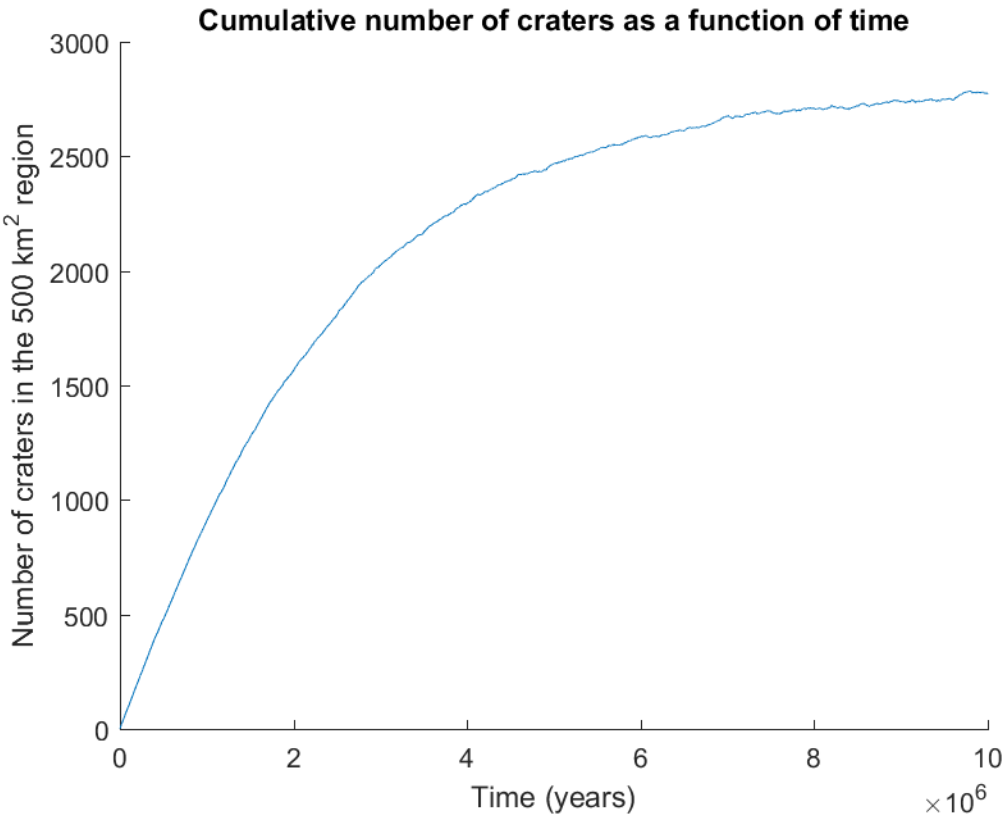
vi. The crater field can be assumed to be saturated when the cumulative number of craters (see part c.) appears to stop decreasing. This seems like a safe assumption, as saturation is inherently defined by the maximum number of craters a field can hold (which is represented graphically by the cumulative number of craters approaching a constant value).

b. <u>Snapshots of the crater fields during simulation</u>



These plots graphically depict the field of craters at four different times during the course of the simulation.

c. Cumulative number of craters on field as a function of time



By qualitatively examining the plot, we can estimate that the time taken to saturate the field was around 7,000,000 years.

d.

i. Crater field plots: The four plots of the crater fields turned out precisely as you'd expect. All the craters have constant sizes and random locations. My four plots depict the field after ¼ of the simulation timespan has elapsed, after ½ has elapsed, after ¾, and then again after the simulation is complete. As expected, the field becomes a mess as it approaches its saturation density. Even after just 5,000,000 years, the field appears to nearly saturated, and when the simulation is complete after 10,000,000 years have elapsed, it's very definitely saturated. We see this reflected in the plot of cumulative number of craters; the number of craters has come to a near-standstill after just 6,000,000 years.

ii. Cumulative crater count: The plot of cumulative craters at a given time plotted against time elapsed turned out just as I expected. Initially, the crater count increases linearly with time; while there are few craters, few to none are being erased and thus the crater count increases. As the density of craters increases, it becomes more and more likely for new craters to land on top of old craters, thereby erasing a few every now and then. As time goes on, the crater density increases and the number of

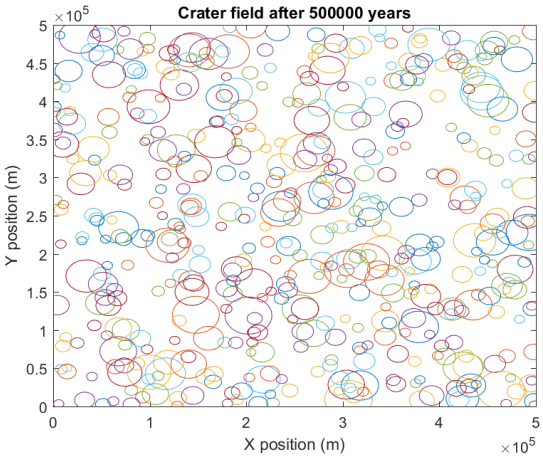
craters erased at each moment in time approaches the number of craters being created. I'm quite confident that the smoothness of this plot is a result of my assumption that every crater has a diameter of 10 kilometers. In general, a 10 kilometer crater won't erase (many) more than one crater, so there aren't likely to be any large dips in the cumulative crater count. Conversely, in a situation where there are a mix of large and small craters, one might imagine a large crater obliterating multiple small craters, potentially creating noticeable dips in the cumulative crater count. That's not the case here, and thus we have a fairly smooth curve. Ultimately, as the slope of the curve decreases and approaches zero, we can get very close to the saturation density. By averaging the crater counts taken during the last tenth of the graph, we arrive at a saturation density of about 2750 craters per 500 km^2.

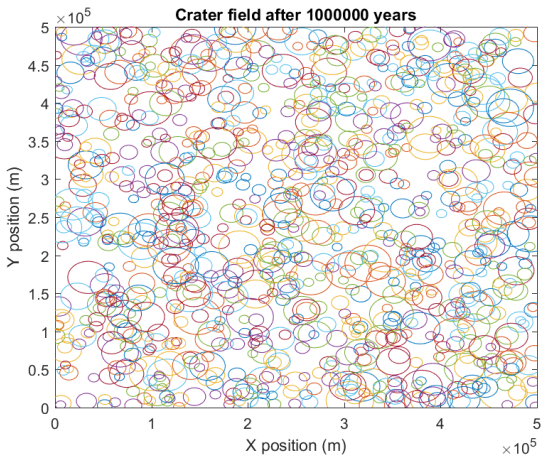
---

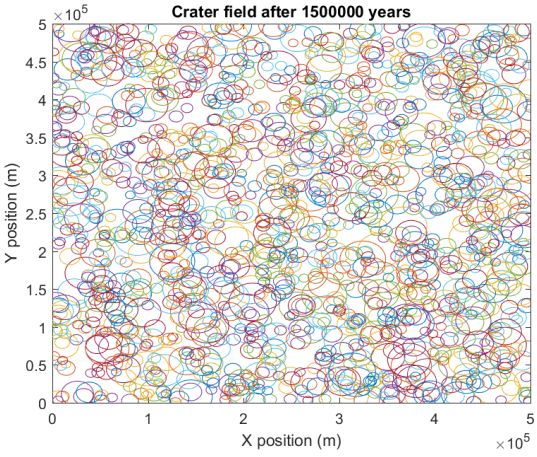
2. Varying Crater Size by Power Law Distribution

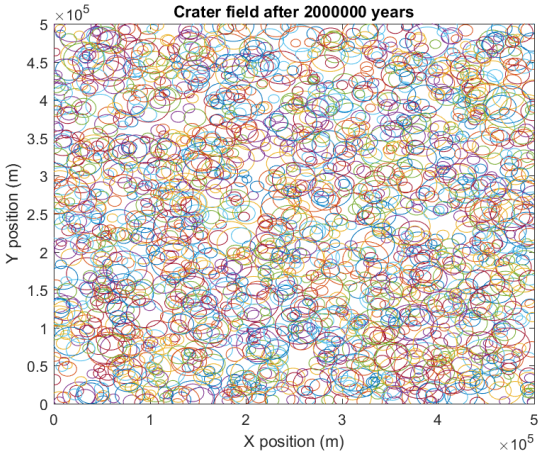
The assumption that I'm primarily changing for part 2 is that the crater size will no longer always be 10 kilometers. Instead, I'm going to apply the power law size frequency distribution (the process by which I accomplished this is described in part 2 of the Appendix). However, doing so requires that I make several new assumptions. First, I have to assume that there's a minimum and maximum possible crater diameter (I assume the minimum to be 10 kilometers and the maximum to be 50 kilometers). I feel quite safe about assuming a maximum diameter of 50 kilometers, as the power law dictates that very few craters larger than this would be created even if I allowed it. I feel much less safe about setting a minimum of 10 kilometers, as we're essentially ignoring the presence of an enormous amount of smaller craters which could easily fit into gaps between larger craters and strongly influence the cumulative crater counts. However, I'm not going to consider them as doing so becomes very computationally expensive (and the instructions said to only consider craters larger than 10 kilometers anyway). I'm also going to assume, in regard to the size frequency distribution n_{cum} (again, seen in part 2 of the Appendix) that the value of the constant b is 2. As far as we've seen, 2 tends to be a typical value (and I believe the math actually requires b to be 2 from a theoretical perspective. It's not much of an assumption, but I should also state that I am assuming that actual craters follow a power law distribution. Despite some deviations (as are always found in experimental settings), I think it's very clear that such a distribution is infinitely more accurate than holding crate size constant. Despite having to make these new assumptions, I think it's very obvious that making crater size vary according to the power law distribution is absolutely worth it considering that a constant crater size was one of my worst initial assumptions.

a. <u>Snapshots of crater fields during simulation</u>

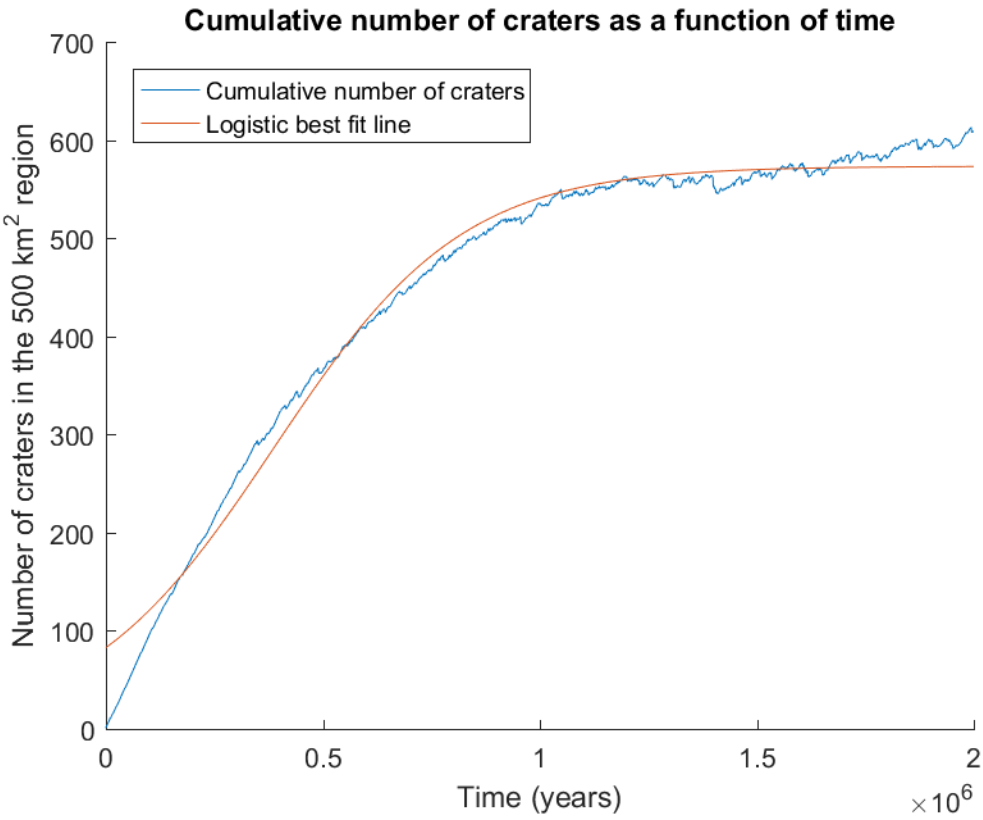

Crater field after 500000 years
×10^5
Y position (m)
X position (m)
×10^5


Crater field after 1000000 years
×10^5
Y position (m)
X position (m)
×10^5


Crater field after 1500000 years
×10^5
Y position (m)
X position (m)
×10^5


Crater field after 2000000 years
×10^5
Y position (m)
X position (m)
×10^5

b. Cumulative number of craters on field as a function of time



Because the variable crater size causes the slope to get a bit messy, I plotted a best fit logistic growth curve alongside the raw data[1]. The slope approaches zero shortly after one million years have elapsed; after this point, the region is saturated, and we find (again, from the best fit line) that the saturation density is approximately 570 craters per 500 square kilometers. As a check, we can trivially calculate an approximation of the average radius of the craters saturating the field by dividing the area by the number of craters, which gives us the average crater area, and then calculate the radius by acknowledging that craters are circles. This yields an average crater diameter of approximately 23 kilometers, which seems relatively reasonable considering that the minimum and maximum possible diameters are 10 and 50 kilometers respectively. Additionally, from looking at the best fit function, we can qualitatively estimate that the field reached saturation after about 1.3 million years.

c. There are three dramatic differences between the above plot and the analogous plot in part 1 in which we assumed constant crater size.
First, the field reached its saturation density much more quickly (after only 1.3 million years) than when all the craters were 10 kilometers across (when it took roughly seven million years). This can be entirely attributed to the larger crater size. The larger the average crater, the fewer craters it will take to fill/saturate

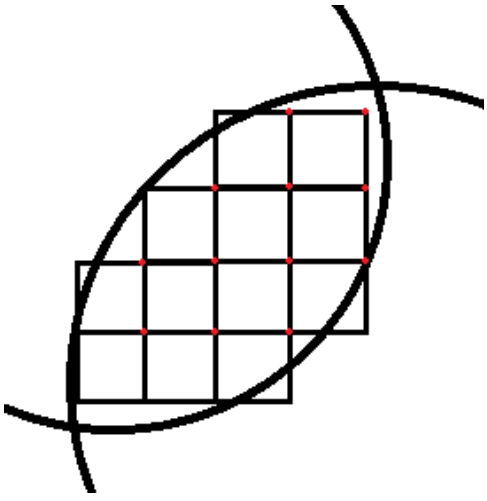
the field. Because we're putting down craters at the same rate as before, but the craters tend to be quite a bit larger, the field is filled more quickly.

Second, the saturation density is now far lower than it was when crater diameter was held at 10 kilometers. This makes sense. Let's estimate that on average, craters now have diameters of 20 kilometers; this corresponds to an area of about 300 km^2. By contrast, craters with diameters of only 10 kilometers have an area of a little less than 100 km^2. It makes sense, therefore, that the saturation density will be on the order of three times higher in the case where we hold crater diameter at 10 kilometers. As we can see from the two plots of cumulative crater count for the two cases, my 'factor of three' assertion appears to hold up reasonably well; the saturation density from part 1 is around four times the saturation density calculated in part 2.

The third difference between the plots is that where the constant crater diameter plot was very smooth, this one is much more jerky and frequently veers down. This is also attributable to larger craters. As I mentioned earlier, larger craters have the potential to instantaneously obliterate multiple smaller craters, causing an immediate drop in the cumulative number of craters, and that's precisely what we can see happened above. As a numerical example, because the largest possible craters have diameters of 50 kilometers and the smallest have diameters of 10 kilometers, a large crater could wipe out on the order of 25 small craters. After approximately 1,450,000 years, one such dip of what looks like about 10 craters happens instantaneously.

The plots of the crater field are also as expected. Where before the craters all had the same diameter, the craters in the new plots vary quite significantly. As I programmed them to, they seem to vary from about 10 kilometers to 50 kilometers with many more small craters than large craters. By the final snapshot, which was taken after 2,000,000 years, the field qualitatively appears to have reached saturation density.

Appendix for Part 1



The above is a simple graphical demonstration of the method by which my program calculates the overlapping areas between circles. It loops through every few hundred points in the crater being covered up and marks those points that are in both craters; these points are marked by red dots. In ultimately calculating the area, I take the number of points the program found in the overlap and extend each of those points out in the *x* and *y* directions for however many points we skipped over earlier. This yields an approximation of the overlapping area. Simply calculating the overlapping area as a magnitude rather than counting individual pixels would have been far simpler and more efficient, but a special case required that we keep track of the specific regions of craters that have been erased: if a crater B partially lands on crater A but doesn't erase it, and crater C then lands on both crater A and B, we need to know exactly what parts of all three craters overlap; crater C should erase this triple overlap region from crater B's area, but not from crater A's area, as crater B already erased that portion of it, so that region must be excluded from the erasure procedure associated with crater C.

Appendix for Part 2

Incorporating a variable crater size into my code was fairly straightforward. Most of my work was done on paper and is not represented in my code, so therefore I'll replicate it here. I began by considering the typical size frequency distribution for craters.

$$n_{cum} = cD^{-2}$$

We were given that a single 10 kilometer crater will be created every thousand years. I should note that under my implementation of part 2, a single crater is created every thousand years with a size that could be anywhere between a minimum of 10 and a maximum of 50 kilometers (which I assume to be the maximum possible crater diameter), though because the average crater size will be much closer to 10 kilometers than it will be to 50, I feel relatively safe about simply assuming that a 10 kilometer crater will be created every thousand years and applying it to the size frequency distribution as follows.

$$1 = c * (1 * 10^4\ m)^{-2}$$

From this, we can solve for the constant c, which will equal $1 * 10^8$.

The next step is to find a method for determining the size of a crater. It should vary between 10 kilometers and my assumed maximum crater size, 50 kilometers, but clearly, my algorithm should generate smaller craters far more frequently than it should larger craters.

First, I integrated the size frequency distribution from the minimum possible crater diameter to the maximum.

$$A = c\int_{1*10^4\ m}^{5*10^4\ m} D^{-2}dD$$

My code then chooses a random number, Z, between 0 and A. Next, I'm going to assert that we want to again integrate n_{cum} from the minimum crater diameter until the integration arrives at Z. The upper bound of this integration, D_{rand}, will be the weighted "random" diameter of the crater. Ergo,

$$Z = 1 * 10^8 \int_{1*10^4\ m}^{D_{rand}} D^{-2}dD$$

By algebraically solving for D_1, we arrive at the following equation, which appears in *crater_gen.m*.

$$D_{rand} = \frac{-c}{Z - {}^{c}/_{1*10^4}}$$

By having MATLAB perform the above integrations and calculations every time a new crater is created and setting that crater's diameter to be D_1, the resulting size frequency distribution closely matches the theoretical distribution determined by n_{cum}. As a check to determine if this method was producing reasonable results, I generated several hundred craters and simply graphed them as a function of their diameter. Sure enough, there was a heavy concentration of smaller craters alongside only a few larger craters.

Code (original source code will be attached)

main.m

```
minD=1e4; % The minimum diameter with which craters can be generated (m)
maxD=5e4; % The maximum diameter with which craters can be generated (m)
c=1e8; % A constant value used to determine the size frequency distribution
b=2; % A constant value used to determine the size frequency distribution
n=@(D_symbolic) c.*D_symbolic.^(-b); % The size frequency distribution of
craters
A=double(integral(n,minD,maxD)); % The integrated area under the size
frequency distribution from the minimum to maximum crater diameters.  This
will be used in crater_gen.m to weight the creation of craters toward smaller
craters.

t_arr=[]; % An array of times, the final entry of which always describes how
much time has passed since the simulation began.  For example, on the first
iteration of the program, t_arr=[0]. On the second, t_arr=[0 1000], and so
on.
n_arr=[]; % An array that describes the cumulative number of craters at a
time, i.e. n_arr(i) describes the number of craters after t_arr(i) years
where i is an arbitrary number less than or equal to the length of t_arr[].
```

```
crat_arr=[]; % An array which will contain every crater that impacts the
surface, where 'crater' is a used-defined class that can record a crater's
relevant properties.

t=0; % The current time.  Each iteration of the while() loop below increases
t until it reaches sim_timespan.
sim_timespan=1E7; % The desired length of our simulation measured in years.
while(t<=sim_timespan+1) % Run this loop while we're within the desired
range.
    %% Crater generation
    crat=crater_gen(A,c,minD); % Create a crater, crat, which now contains
the properties dsecribed in crater.m.
    crater_plotter(crat,t,sim_timespan); % Plot the crater field at this
moment in time, t.
    crat_arr=[crat_arr; crat]; % Add the crater we just created to the array
of all craters.
    t_arr=[t_arr; t]; % Add the current time to the list of elapsed times.

    %% Overlap analysis
    % The next goal is to consider every crater surrounding the crater we
    % just created (i.e. we must compare crat to every other crater in
    % crat_arr).  At each crater, crat_arr(i), we must determine if it
    % overlaps with our new crater, crat.  If so, we must THEN consider
    % whether the region of crat_arr(i) that is being occluded by crat has
    % already been erased.  The parts that have not already been erased
    % should now be erased by the overlapping region with crat.  The parts
    % that were previously erased should be ignored.
    for i=1:length(crat_arr)-1 % Our new crater, crat, is now the final entry
of crat_arr, so we must consider every crater except for the last one.
        R1=crat.R;
        R2=crat_arr(i).R;
        d_x=crat.x_center-crat_arr(i).x_center; % Calculate the distance
between the centers of crat and crat_arr(i) in the x direction.
        d_y=crat.y_center-crat_arr(i).y_center; % Calculate the distance
between the centers of crat and crat_arr(i) in the y direction.
        d_mag=sqrt(d_x^2+d_y^2); % Calculate the magitude of distance between
the centers of crat and crat_arr(i).
        A_intersect=real((R1^2)*acos((d_mag^2+R1^2-
R2^2)/(2*d_mag*R1))+(R2^2)*acos((d_mag^2+R2^2-R1^2)/(2*d_mag*R2))-0.5*sqrt((-
d_mag+R1+R2)*(d_mag+R1-R2)*(d_mag-R1+R2)*(d_mag+R1+R2)));
        % A_intersect calculates the overlapping are between crat and
        % crat_arr(i).  If crat and crat_arr(i) are not overlapping,
        % A_intersect goes to 0.  If crat is smaller than crat_arr(i) and
        % lands entirely within crat_arr(i)'s area, A_intersect will equal
        % crat's area.

        % Below, we will determine whether the region of crat_arr(i) being
        % occluded by crat was previously erased.  If it was, we will
        % ignore it.  If it wasn't, we will mark that region to be erased.
        % Ideally, this process would be carried out by recording every
        % (x,y) pair in the region to be erased and associating every one
        % of those pairs with crat_arr(i), i.e. crat_arr(i).x_erased and
        % crat_arr(i).y_erased would store an enormous number of (x,y)
        % pairs that would denote 1 m^2 regions within crat_arr(i) that
        % have been erased (and at the end of the day, we would sum up the
        % areas described by those pairs and subtract that area from
        % crat_arr(i)'s area before the operation.
```

```
        % The implementation below is essentially the process that is
        % described above, though with one modification made for the sake
        % of efficiency: Instead of examining every single (x,y) point in
        % the overlapping region, I only examine one in every few hundred
        % (x,y) points.  The number of points that we skip over is defined
        % by the constant accu_fact, which I have set to 500 for part 1 and
        % 1000 for part 2. To calculate the area to be erased, we will
        % multiply the number of points we just counted by accu_factor^2.
        % Throughout the rest of this documentation, I will refer to number
        % of pixels over which we skip as accu_fact; keep in mind that it
        % typically denotes either 500 meters or 800 meters.
        dots=0; % The number of accu_fact m^2 regions that crat has just
erased from crat_arr(i).
        accu_fact=500; % The number of 1 m^2 regions over which to skip
during our analysis.
        if (A_intersect>0 & crat_arr(i).exist==1) % We only consider nonzero
overlaps between craters, and we additionally make sure that crat_arr(i) has
not already been erased.
        %% Point counting
        % We will now consider every few hundred (accu_fact) points in
        % crat_arr(i) and a) determine which one of these points are within
        % the overlapping region with crat and b) exclude regions in
        % crat_arr(i) that have already been erased.
            for y=crat_arr(i).y_center-
crat_arr(i).R:accu_fact:crat_arr(i).y_center+crat_arr(i).R % Loop through the
rows in crat_arr(i).
                for x=crat_arr(i).x_center-
crat_arr(i).R:accu_fact:crat_arr(i).x_center+crat_arr(i).R % Loop through the
columns in a given row.
                    % Keep in mind that at this level, we're examining a
                    % single point (x,y) in crat_arr(i) and determining
                    % whether it should be erased.
                    already_erased=0; % The point (x,y) in question has not
yet been erased.
                    d_mag_crat=sqrt((crat.x_center-x)^2+(crat.y_center-y)^2);
% The distance between (x,y) and the center of crat.
                    d_mag_crat_i=sqrt((crat_arr(i).x_center-
x)^2+(crat_arr(i).y_center-y)^2); % The distance between (x,y) and the center
of crat_arr(i).
                    if (d_mag_crat<crat.R & d_mag_crat_i<crat_arr(i).R) % If
the two magnitudal distances are less the radii of the respective craters,
(x,y) is within the overlapping region.
                        % Because craters are assigned completely random
                        % coordinates on the plane and because how we're
                        % looping through all of the (x,y) pairs within
                        % those craters, we have to round the (x,y) pair
                        % we're examining to the nearest 'clean' number.
                        % This is required because part of this process is
                        % checking whether the (x,y) pair in crat_arr(i)
                        % that we're examining has already been erased.
                        x_pos=round(x/accu_fact)*accu_fact; % Round the x
position in question to the nearest accu_fact
                        y_pos=round(y/accu_fact)*accu_fact; % Round the y
position in question to the nearest accu_fact
                        for j=1:length(crat_arr(i).x_erased) % Check to see
if (x_pos,y_pos) were already erased from crat_arr(i).
                            % Here, j refers to the jth item in the matrix
```

```
                            % of (x,y) pairs that have already been erased
                            % from crat_arr(i).
                            if (crat_arr(i).x_erased(j)==x_pos &
crat_arr(i).y_erased(j)==y_pos)
                                % If we find that (x_pos,y_pos) already
                                % exist in the matrix of erased (x,y) pairs
                                % for crat_arr(i), we mark it as such so
                                % that we won't attempt to erase it again
                                % later.
                                already_erased=1;
                            end
                        end
                        if already_erased==0 % If (x_pos,y_pos) wasn't found
in the matrix of (x,y) pairs erased from crat_arr(i), the point in question
wasn't already erased.  We will now erase it.
                            crat_arr(i).x_erased=[crat_arr(i).x_erased
x_pos]; % Add x_pos to the matrix of erased x positions.
                            crat_arr(i).y_erased=[crat_arr(i).y_erased
y_pos]; % Add y_pos to the matrix of erased x positions.
                            dots=dots+1; % We have located a point that must
be erased; 'dots' records this.
                        end
                    end
                end
            end
        end
        %% Erased area subtraction
        % If A_intersect between crat and crat_arr(i) equals zero, the
        % area to be erased below will also equal zero (because 'dots' will
        % equal zero, thereby causing no area to be subtracted from
        % crat_arr(i). If A_intersect>0, then the erased area will not
        % equal zero and we will subtract a nonzero area from the remaining
        % area of crat_arr(i).
        area_erased=dots*(accu_fact^2); % The number of square meters that
have just been erased from crat_arr(i).
        crat_arr(i).area_r=crat_arr(i).area_r-area_erased; % The remaining
area in crat_arr(i).

        % If the remaining area of crat_arr(i) falls below half of its
        % initial area, we consider it to have been erased.
        if (crat_arr(i).area_r/crat_arr(i).area_i<0.5 & crat_arr(i).exist==1)
% We also verify that the crater in question still exists.
            crat_arr(i).exist=0; % We mark crat_arr(i) as having been erased.
        end
    end

    %% Crater counting at time t
    % We're done comparing crat to all other craters in crat_arr. We must
    % now count the total number of craters at this iteration.  One has
    % been added from the previous iteration (in the form of crat), but one
    % (or more) may have been erased.
    n_temp=0; % Initial count of craters on this iteration.
    for i=1:length(crat_arr)
        if crat_arr(i).exist==1 % Only consider craters that haven't been
erased.
            n_temp=n_temp+1; % Iterate upwards, resulting in the total number
of craters at this moment.
```

```
        end
    end
    n_arr=[n_arr; n_temp]; % Append the latest crater count to the array of
counts. Keep in mind that n_arr(i) corresponds to t_arr(i).

    disp([num2str(100*(t/sim_timespan)) '% done. Crater count: '
num2str(n_arr(length(n_arr)))]); % A simple progress indicator.
    t=t+1000; % At the end of each iteration of the while() loop, we step
forward in time 1000 years.
end

%% Plot the cumulative number of craters vs. time.
figure;
hold on;
plot(t_arr,n_arr); % Plot the matrix of times against the matrix of
corresponding crater counts.
title('Cumulative number of craters as a function of time');
xlabel('Time (years)'); % Label the x axis.
ylabel('Number of craters in the 500 km^2 region'); % Label the y axis.
saveas(gcf,['crater_count.png']); % Save a copy of the plot.
```

crater.m

```
classdef crater
    properties
        D; % The diameter of the crater.
        R; % The radius of the crater.
        x_center; % The random x position where the center of the crater will
be placed.
        y_center; % The random y position where the center of the crater will
be placed.
        area_i; % The initial area of the crater, as calculated from the
radius, R.
        area_r; % The remaining area of the crater.  At first, it will be set
equal to the crater's initial area, and as the crater is covered up by other
craters, it will be subtracted from.  When it equals less than half of the
initial area, the crater will be marked as erased.
        x_erased=[]; % A matrix of x positions within the crater that have
been erased.
        y_erased=[]; % A corresponding matrix of y positions within the
crater that have been erased.
        exist; % A flag that denotes whether the crater exists.  exist=1
denotes that the crater exists, exist=0 denotes that it has been erased.
    end
end
```

crater_gen.m

```
function [crat] = crater_gen(A,c,minD)
    crat=crater; % Generate a new crater of class 'crater'.
    Z=randi([0 uint32(A)],1,1); % We choose a completely random area Z that
is a subset of A (A being the integral of the size frequency distribution
from the minimum possible crater diameter to the maximum.
    % Now, through analytic integration, we find a crater diameter D1 such
    % that Z=integral(n_cum) from D=minD to D=D1.
    %crat.D=-c/(Z-(c/minD)); % The analytic solution to the integration of
the size frequency distribution, algebraically solved for the upper limit of
integration, crater diameter D.
```

```
    crat.D=1e4;
    crat.R=crat.D/2; % The radius of the crater.
    crat.x_center=randi([0 5e5],1,1); % A random x coordinate at which to
place the crater.
    crat.y_center=randi([0 5e5],1,1); % A random y coordinate at which to
place the crater.
    crat.exist=1; % The crater is initially flagged as existing.
    crat.area_i=pi*(crat.R^2); % The initial area of the crater.
    crat.area_r=crat.area_i; % The remaining area of the crater, at first set
as the crater's initial area.  To be whittled down as other craters land on
top.
end
```

crater_plotter.m

```
function [] = crater_plotter(crat,t,sim_timespan)
    theta=linspace(0,2*pi);
    plot(crat.x_center+crat.R*cos(theta),crat.y_center+crat.R*sin(theta)); %
Plot the newly created crater, crat.
    xlabel('X position (m)'); % Label the x axis.
    ylabel('Y position (m)'); % label the y axis.
    title(['Crater field after ' num2str(uint32(t)) ' years']); % Set a
title.
    hold on; % We'll keep the same figure throughout the entire simulation
and just add a new crater to it as each iteration is executed (rather than
redrawing the entire field every time a crater is added).
    axis([0 5e5 0 5e5]); % Set the axis to 500 km by 500 km.
    % Save a few snapshots of the crater field after the simulation is 1/4,
    % 1/2, 3/4, and 1/1 complete.
    if (t==sim_timespan*0.25 | t==sim_timespan*0.5 | t==sim_timespan*0.75 |
t==sim_timespan)
        saveas(gcf,[num2str(t) '_years.png']);
    end
end
```

Bibliography

1. Credits for the best fit for a logistic curve go to James Conder: https://www.mathworks.com/matlabcentral/fileexchange/41781-fit-logistic-t-q-