Cytoskeleton Polymerization Dynamics

Part I. Spreading the butter: Integrating the Chemical Master Equation.
In part I, we will write a script that uses the chemical master equation to describe the time-evolution of a growing and shrinking cytoskeletal filament. We will use this script to predict a steady-state probability distribution of filament lengths.

Part II. Filament Image Analysis.
In part II, we will write a script to measure filament length from images of actin filaments. We can then compare our measured distribution of filament lengths to our prediction from Part I.

Part I. Spreading the Butter: Integrating the Chemical Master Equation
In this script, we will learn a bit about cytoskeletal filaments, how to write a numerical integrator in MATLAB, and the principle of the master equation.

The Chemical Master Equation
We will oftentimes be interested in how probability distributions of some process (such gene expression levels, or lengths of polymers) evolve over time. We can capture our understanding of these systems by writing differential equations, which will more often than not be relatively hairy to solve analytically. One way we can write out these differential equations is by writing what is called a master equation. In general, a master equation will have the form

\[ \text{probability at time } t + \Delta t = \text{probability at time } t + \text{probability added} - \text{probability lost} \]

As an example, let’s think of how we can compute the probability of having a filament with a length \( \ell = 3 \). There are a total of four things that can happen:

1. A filament of length \( \ell = 2 \) grows by one monomer to become \( \ell = 3 \).
2. A filament of length \( \ell = 4 \) degrades by one length to become \( \ell = 3 \).
3. A filament of length \( \ell = 3 \) can degrade by one to become \( \ell = 2 \).
4. A filament of length \( \ell = 3 \) or grow by one monomer to become \( \ell = 4 \).

With this knowledge of how we can build a filament, we can write the complete master equation for a filament of length \( \ell = 3 \) as

\[ P(\ell = 3, t + \Delta t) = P(\ell = 3, t) + r \Delta t P(\ell = 2, t) + \gamma \Delta t P(\ell = 4, t) - \Delta t P(\ell = 3, t)(r + \gamma). \]

Rather than doing this by hand for all possible lengths from \( \ell \in [0, \infty) \), we can write the general master equation as
With this master equation in place, we can now figure out how the distribution changes with time! To do so, we’ll use the method of **Forward-Euler integration**.

**Forward-Euler Integration**

Developing simple ways to solve ordinary differential equations has long been an area of intense research. While deriving the analytical solution may be simple in some cases, it is often useful to solve them numerically, especially if slamming out the analytical solution will give you carpal tunnel.

While there are many ways to numerically integrate these equations, in this tutorial we will examine the **Forward Euler method**. Say we have an ordinary differential equation such as

\[
\frac{dx}{dt} = -\frac{x}{\tau},
\]

where \(\tau\) is some time constant and \(t\) is time. Rather than solving this analytically (although it is trivial), we can solve it numerically by starting at some given value of \(x\), evaluating this equation for a given time step \(\Delta t\), and updating the new value of \(x\) at this new time \(t + \Delta t\). We can phrase this mathematically as

\[
x(t + \Delta t) = x(t) - \frac{x(t)}{\tau} \Delta t.
\]

Say our initial value (\(x\) at \(t = 0\)) is \(x = 10\) and \(\tau = 1\). We can take a time step \(\Delta t = 0.1\) and find that the change in value of \(x\) is

\[
\Delta x = -\frac{x}{\tau} \Delta t = -1.
\]

We can then compute the new value of \(x\) at time \(t + \Delta t\) as

\[
x(t + \Delta t) = x(t) + \Delta x = 10 - 1 = 9.
\]

We can then take another step forward in time and repeat the process for as long as we would like. As the total time we’d like to integrate over becomes large, it becomes obvious why using a computer is a far more attractive approach than scribbling it by hand.

A major point to be wary of is the instability of this method. The error in this scales with the square of our step size. We must choose a sufficiently small step in time such that at most only one computable event must occur. For example, if we are integrating exponential growth of bacterial cells, we don’t want to take time steps larger than a cell division! This requirement is known as the **Courant-Friedrichs-Lewy condition** and is important for many different time-marching computer simulations.

**Integrating the Chemical Master Equation**

Let’s write an integrator to numerically solve this master equation. We’ll start by defining the chemical parameters of the system, \(r\) and \(\gamma\).

```plaintext
% Define the parameters.
% r = 9; % on rate of a monomer to the filament. Dimensions are inverse time
```

\[
P(\ell, t + \Delta t) = P(\ell, t) + r\Delta tP(\ell - 1, t) + \gamma\Delta tP(\ell + 1, t) - \Delta tP(\ell, t)(r + \gamma).
\]
gamma = 10; % off rate of a monomer to the filament. Dimensions are inverse time

While the length of our cytoskeletal filament can be essentially infinite, it’s not really possible for us to compute that. To get around this we can set what a "maximum length" should be in our simulation integrate over that range. You should note that this means our solution will not be exact, but is good enough for what we are after. For more exact numerical calculations, you should consider looking into MATLAB's standard ODE solvers.

We will be integrating essentially over two dimensions – filament length and time. This means that we will want to set up a two dimensional matrix where each column will represent a time point and each row will be an integer length of our filament. Let's go a head and define this matrix as well as the other parameters of our system.

% Define the integration parameters.
maxLength = 100; % The maximum length of the filament that we will integrate over.
totalTime = 50; % Total time of the integration in dimensions of time.
dt = 1/ 50; % Time step for the simulation. Note that this needs to be sufficiently small!
timeSteps = totalTime / dt; % The total number of time steps we will be taking.

% Set up the matrix in which we will store the values.
P = zeros(maxLength, timeSteps); % A two-dimensional matrix full of zeros.

Before we can begin the integration, we have to set the initial condition (our matrix is full of zeros after all!). We could start our integration in any way, but for fun, let's set the initial condition for there to be zero filaments. This means we will set the first index of our matrix.

% Set the initial condition.
P(1, 1) = 1.0; % Remember that MATLAB indexing begins at 1.

Now we are finally ready to begin the integration. However, we have to be cognizant of one issue – the boundary conditions. When we have $\ell = 0$, for example, it doesn't make sense to have a degradation term $\gamma \Delta t P(\ell, t)$ because 0 can't be degraded! Likewise, you can't have a production term $r \Delta t P(\ell - 1, t)$. Similarly, for $\ell = \text{maxLength}$, we cannot have a monomer incorporation event to create a polymer of length $\ell = \text{maxLength} + 1$. Finally, we cannot have filaments of length $\ell = \text{maxLength} + 1$, which means that we cannot have a monomer loss event to create $\ell = \text{maxLength}$. When we begin our integration, we will have to compute the probabilities at the boundaries $\ell = 0$ and $\ell = \text{maxLength}$ individually. Let's go ahead and perform the integration.

% Loop first through each time point. We'll start at time t=2 since we set the % initial condition.
for t=2:timeSteps
    % We'll compute the initial condition, when ell = 1, which is our first condition.
    ell = 1;
P(ell, t) = P(ell, t-1) + gamma * dt * P(ell + 1, t-1) - r * dt * P(ell, t - 1);

    % Now we can deal with the other boundary condition, when ell = maxLength.
    ell = maxLength;
P(ell, t) = P(ell, t-1) + r * dt * P(ell-1, t-1) - gamma * dt * P(ell, t-1);
% The boundary conditions are satisfied, so we can jump ahead and go through the rest of the lengths of the filament.
for ell=2:(maxLength - 1) % -1 so we don't compute the final length!
P(ell, t) = P(ell, t-1) + r * dt * P(ell-1, t-1) + gamma * dt * P(ell+1, t-1) - ...
dt * P(ell, t-1) * (r + gamma);
end % This ends the length loop.
end % This ends the time loop.

And that's it! In just a few lines of code, we were able to integrate our master equation! Let's take a look at the distribution. Since we have three axes of information (length, time, and probability), we can plot it as a 3-D bar plot, where the height of the bars will be representative of the probability of that length, the x-dimension will be length, and the y-dimension will be time steps. Since we have so many time steps, we'll only plot every twentieth time point.

% Plot the probability vector as 3-D bar plot.
bar3(P(1:40, 1:70:timeSteps)); % 1:20:timeSteps plots the first bar to the last bar taking steps of 20
% Now we just have to add labels like good scientists.
ylabel('length in monomers')
xlabel('time steps x 20')
zlabel('P(l, t)');
% Set a specific angle for viewing.
view([52.900 35.600])
As we rotate this plot around, we can see that it spreads out over time, with the mode being at 0. When we look at the end of the simulation, the distribution does look qualitatively exponential. Rather than saying just what it looks like, we can actually solve for this distribution analytically.

Above, we wrote out the master equation that describes the length of the distribution. We can analytically solve for the steady-state distribution using:

We can rewrite this in the form of an ordinary differential equation which summarizes the change in the probability length distribution over time,

$$\frac{dP(\ell, t)}{dt} = rP(\ell - 1) + \gamma P(\ell + 1) - P(\ell)(r + \gamma).$$

At steady state, the change in probability for a given length $\ell$ over a time step $dt$ is 0. By setting the above equation equal to zero, we can solve for this probability distribution,

$$\frac{dP(\ell)}{dt} = 0 = rP(\ell - 1) + \gamma P(\ell + 1) + P(\ell)(r + \gamma)$$

$$\Rightarrow P(\ell)(r + \gamma) = rP(\ell - 1) + \gamma P(\ell + 1).$$

Now, that’s not very informative of what the distribution should be because the distribution we are interested in $P(\ell)$ is dependent on two other lengths, $\ell + 1$ and $\ell - 1$. We can solve for this distribution $P(\ell)$ as a function of the reference length $P(\ell = 0)$ using the principle of detailed balance.

**A Brief Aside: Detailed Balance**

The principle of detailed balance states that there is no such thing as an irreversible process and at equilibrium, each elementary elementary process is equilibrated by its reverse process. For example, let’s imagine we have an irreversible process of the form

$$A \rightarrow B.$$  

The principle of detailed balance states that at equilibrium, each elementary transition is in equilibrium,

$$A \leftrightharpoons B.$$  

If the forward transition $A \rightarrow B$ occurs at a rate $r_1$ and $B \rightarrow A$ occurs with a rate $r_2$, we can write an expression for $B$ in steady state in terms of $A$,

$$r_1 A = r_2 B \Rightarrow B = \frac{r_1}{r_2} A.$$  

With this principle in hand, we can solve for our length distribution.

**Solving for the distribution using detailed balance**

A few lines ago, we wrote an expression for $P(\ell)$ in terms of $P(\ell - 1)$ and $P(\ell + 1)$. We can solve this generally by realizing that through the principle of detailed balance,
\[ rP(\ell) = \gamma P(\ell + 1). \]

As an example, we can look at the case of \( P(\ell = 1) \) and solve it in terms of \( P(\ell = 0) \).

\[ rP(0) = \gamma P(1). \]

We can solve for \( P(1) \) as

\[ P(1) = \frac{\ell P(0)}{\gamma}. \]

Great! Now, let's try solving for \( P(\ell = 2) \). We can write

\[ rP(1) = \gamma P(2). \]

Using our expression for \( P(1) \) derived above, we can rewrite this as

\[ r\frac{\ell P(0)}{\gamma} = \gamma P(2). \]

Solving for \( P(2) \) yields

\[ P(2) = \frac{\ell \times \ell P(0)}{\gamma} = \left( \frac{\ell}{\gamma} \right)^2 P(2). \]

Of course, we probably don't want to do this over and over again. A keen eye will notice the pattern that this should simplify to the general form for \( P(\ell) \) in terms of \( P(0) \) as

\[ P(\ell) = \left( \frac{\ell}{\gamma} \right)^\ell P(\ell = 0). \]

Now, we are almost there! Ideally, we would have an expression for \( P(\ell) \) that is independent of any other probability, meaning your function should only be a function of \( r, \gamma \), and \( \ell \). We can get there by realizing that this probability distribution \( P(\ell) \) (like every other probability distribution) should be normalized,

\[ \sum_{\ell=0} P(\ell) = 1. \]

This normalization condition can be rewritten by subbing in our expression for \( P(\ell) \) above,

\[ \sum_{\ell=0} \left( \frac{\ell}{\gamma} \right)^\ell P(\ell = 0) = 1. \]

Since \( P(\ell = 0) \) is constant for all lengths \( \ell \) in the sum, we can pull this term infront of the sum resulting in

\[ P(\ell = 0) \sum_{\ell=0} \left( \frac{\ell}{\gamma} \right)^\ell = 1. \]

While this may look like a pain to solve at first, we can recognize that the summation is exactly the geometric series, which converges to
\[
\sum_{n=0}^{\infty} a^n = \frac{1}{1 - a}
\]

if \( a < 1 \). For a refresher on the convergence of this series, scroll down to the very bottom of this tutorial. Note that the convergence condition demands that \( r < \gamma \). We can use this convergence to now solve for our desired distribution \( P(\ell = 0) \) as

\[
P(\ell = 0) \frac{1}{1 - \frac{r}{\gamma}} = 1
\]

\[
\Rightarrow P(\ell = 0) = 1 - \frac{r}{\gamma}
\]

Plugging this result into our expression for \( P(\ell) \), we arrive at the analytical solution for the steady state distribution,

\[
P(\ell) = \left( \frac{\ell}{\gamma} \right) \left( 1 - \frac{r}{\gamma} \right) e^{-\ell \log(\gamma/r)},
\]

which is an exponential distribution! To see if our numerical integration matches our analytical solution for the steady state distribution, we can simply plot them on top of each other.

```matlab
% Set up a length vector
lengths = 0:1:maxLength-1;
% Set up a more finely spaced array of points to compute the analytical solutions.
L = linspace(0, maxLength, 1000);

% Compute the analytical solution.
soln = (r / gamma).^L * (1 - (r / gamma));

% Plot the last time point of our integration and the analytical solution.
bar(lengths, P(:, timeSteps), 'DisplayName', 'Numerical Solution')
hold on
% Plot the analytical solution.
plot(L, soln, 'r-', 'LineWidth', 2, 'DisplayName', 'Analytical Solution');

% Of course, always labels.
xlabel('length in monomers')
ylabel('probability')
legend('-DynamicLegend')
% Set the limits from 0 to maxLength -1
xlim([0, maxLength])
hold off
```
So, we have shown that our analytical solution matches our numerical solution (nearly) exactly.

**In conclusion**

In this tutorial, we have learned a bit about how to write chemical master equations for biological processes, how to numerically integrate them, and proved the validity of the approach by matching our solution to the analytical solution.

**Extra Stuff!**

**Proving the convergence of the geometric series**

The geometric series is given as

\[ S = \sum_{n=0}^{\infty} a \lambda^n, \]

where \( \lambda < 1 \). We can evaluate the first few terms to arrive at

\[ S = a \lambda^0 + a \lambda^1 + a \lambda^2 + a \lambda^3 \ldots \]

By pulling out one factor of \( \lambda \), this can be rewritten as
By realizing that the final three terms in the parenthesis are exactly what we defined as \( S \), the series becomes
\[
S = \lambda(a\lambda^{-1} + a\lambda^0 + a\lambda^2 + \ldots).
\]
Performing some simplification, we come to
\[
S = \lambda(a\lambda^{-1} + S).
\]
Now you have a tool in your toolbox for when you run across the geometric series.

Part II. Filament Image Analysis.

We'll start by opening up a single image and segmenting filaments (i.e. dividing the image into filaments and background). Then, we'll figure out how measure length for all our filaments. Afterwards, we'll program a for loop to do the same for all the images and produce graphs of the filament length distribution.

% Get info about all the images.
images = dir('*.*tif'); %makes a list of all the .tif files within our current directory
% and saves it in "images".

Check out "images" in your MatLab workspace. See how it contains useful info, like the names of our images?

We will use this info later to loop over all our images, analyzing them all. But for now, we'll open just one.

% Read one of the images into MatLab.
im = imread([images(1).name]); %opens the first name in the "images" list.

% View the image.
imshow(im,[]);
Warning: Image is too big to fit on screen; displaying at 67%

Note that, as provided, the image will only let you measure the length of the filaments in units of pixels. We will want to turn these pixels into $\mu m$ later on. To make this possible, we need to calibrate the image using the image of the calibration grid provided with this dataset. What you will learn from this grid is that the conversion from pixels to $\mu m$ is given by

$\text{Calibration} = 120/1722; \quad \mu m/pixel$

Now, we're ready to start processing the image.

```matlab
% Apply a median filter to reduce noise in the image.
imFilt = medfilt2(im); % sets each pixel to the median intensity value of the 3x3 pixel box centered there.

% Scale image intensity.
imNorm = mat2gray(imFilt); % scales intensities so min = 0 and max = 1.

% Let's look at the resulting image
imshow(imNorm)
```
Segmenting our image.

How can we distinguish filaments from background? For one thing, filaments are bright. We could set a threshold intensity that would separate filaments (brighter than threshold) from background (dimmer than threshold). But how do we know what this threshold value should be?

% Look at the image histogram.
imhist(imNorm, 1000); %histogram of image data. 1000 = number of bins.
The large peak in the histogram corresponds to the background pixels while the bump to the right of the peak corresponds to the pixels within filaments. From here, it looks like a threshold of 0.15 could separate this bump from the large peak. We can also estimate the threshold by using `imshow` and using the "inspect pixel values" tool. Note that you might want to comment out the next line so that a new window of `imtool` isn't opened every time you run this script.

```matlab
%imtool(imNorm)
```

`imtool` and "inspect pixel values" confirms that a threshold of 0.15 could work. Of course, you can adjust that to improve your segmentation.

```matlab
% Try a threshold
thresh = 0.15; %replace this value with your best guess.
imThresh = imNorm > thresh; %keep only pixels above threshold
imshow (imThresh);
```
How does this threshold look? No good? Try a new value for "thresh" and repeat.

Once we're happy with our threshold, let's figure out more ways to remove everything but real filaments from our image.

% Clear the border. We can't correctly measure a filament that extends beyond our image.
imClear = imclearborder(imThresh);   %supresses structures lighter than their surroundings and touching a border.

We also know the approximate size (area) of an individual filament, and we know that it should be skinny in shape. Can we check each object in our image to see whether it has these properties? If an object doesn't satisfy these criteria, let's throw it out.

% Identify objects that might be filaments, and give them each a label.
imLab = bwlabel(imClear); %labels connected components in a 2D binary image.
imshow(imLab);
Check each object, and only keep those that are skinny (large "eccentricity") and a reasonable size (area).

We will use MatLab's super-useful regionprops function to measure object properties. Google "matlab regionprops" to learn more!

```matlab
% Use regionprops to find area and eccentricity of a binary array, imLab.
props = regionprops(imLab, 'Area', 'Eccentricity');
```

Like we did for the pixel intensity threshold for our filament segmentation, we need to commit to values of Area and Eccentricity corresponding to our filaments. The eccentricity of a circle is zero, while that of an ellipse is smaller than 1. As we want structures that are extended and not "blobs", the eccentricity then gives us an opportunity to filter out such blobs. Let's stare at the histogram for the Eccentricity values corresponding to each region in imLab. Here, you need to remember that props is a structured array. If I want to pull all values in the field Eccentricity out of this array, I need to do [props.Eccentricity].

```matlab
histogram([props.Eccentricity])
xlabel('eccentricity')
ylabel('number of regions')
```
We see that an eccentricity threshold of 0.5 might allow us to filter out blobs.

```matlab
minEccentricity=0.5;
```

Now, let's examine the areas

```matlab
histogram([props.Area],[0:20:1500])
xlabel('area (pixels)')
ylabel('number of regions')
```
The decision here is less clear. Clearly, we should filter out regions with too small of an area. One idea would be to filter random speckles. However, maybe more importantly, is to acknowledge that we cannot measure sizes beyond the diffraction limit. The diffraction limit is about 500 nm or 7 pixels as suggested by the Calibration introduced above. A circle of diameter 7 pixels has an area

\[(\frac{0.5}{\text{Calibration}}/2)^2\pi\] %area of a diffraction-limited spot in pixels

\[\text{ans} = 40.4328\]

So, let's choose a slightly larger minimum area

\text{minArea}=100;

\text{approvedObj} = \text{zeros(size(im))}; %we'll put only objects of certain size/shape here.
\text{for i=1:length(props)}
  \text{if props(i).Eccentricity > minEccentricity && props(i).Area > minArea% && props(i).Area < 1000}
    \text{approvedObj} = \text{approvedObj} + \text{(imLab==i)}; %keeps only approved objects from imLab.
  \text{end}
\text{end}

% This is the new image containing only the filtered filaments
\text{imshow(approvedObj)}
% Let's update our image of labeled objects so that it only contains approved objects.
imLab = bwlabel(approvedObj); %relabel only these approved objects.

We have segmented our image so that it contains the objects (filaments) we want to measure! Hooray! Pretty awesome.

**Measuring filament length.**

Now, how can we measure the length of the filaments? We can use some MatLab functions to turn our filament objects into "sticks" just one pixel wide. If we do that, then the area of our sticks will be equal to the filament length.

```
skel = bwmorph(approvedObj, 'skel', Inf); %uses bwmorph to find one-pixel-wide % skeleton of all objects.
spur = bwmorph(skel, 'spur'); %remove undesired short spurs % (default for spur = if one pixel has 0 of 4 possible direct neighbors, remove).
imLab = bwlabel(spur); %relabel these skeletons.

imshow(imLab)
```
% Finally, measure the filament lengths!
% Use regionprops Area to find the area of each object (= length since it's a skeleton!)
props = regionprops(imLab, 'Area');
for i=1:length(props)
    sizes(i) = sum(sum(imLab==i));
end

% Make a histogram to see our measured filament lengths.
histogram(sizes, 200); %200 is the number of bins.
Loop over all images to measure all filaments.
Now that we've figured out our analysis pipeline, we can apply it to lots of images.

```matlab
FilamentLengths = []; images = dir('*.tif'); % makes a list of the .tif files within our current directory.
for i=1 %:length(images)
    % Read the images
    im = imread(images(i).name);
    imFilt = medfilt2(im); %median filter
    imNorm = mat2gray(imFilt); %scales intensity between 0 and 1
    thresh = graythresh(imNorm); %alternatively: find threshold using Otsu's method
    imThresh = imNorm > thresh; %threshold image
    imClear = imclearborder(imThresh); %get rid of stuff touching border
    % Label objects and apply filters based on area and eccentricity.
    imLab = bwlabel(imClear);
    props = regionprops(imLab, 'Area', 'Eccentricity');
    approvedObj = zeros(size(imLab)); %we'll put only objects of certain size/shape here.
    for j=1:length(props)
        if props(j).Eccentricity > 0.5 && props(j).Area > 100 && props(j).Area < 1000
            approvedObj = approvedObj + (imLab==j); %keeps only approved objects from imLab.
        end
    end
end
```
Testing our model’s predictions

Let's plot the results for all images!

```matlab
histogram(FilamentLengths, [0:10:150]) %plot the histogram using bins with edges determined by
xlabel('filament length (pixels')
ylabel('number of filaments')
```

What do you think? Does this look like an exponential distribution as predicted by the simple model of polymerization? Let's look at this histogram using a log-scale y-axis

```matlab
HistHandle=histogram(FilamentLengths, [0:10:150]); %we are saving information about the histogram
xlabel('filament length (pixels')
ylabel('number of filaments')
set(gca,'YScale','log')
```
It does look more or less like a line, doesn't it? We can go one step further and try to fit this distribution to an exponential functional form. To make this possible, we need to remember that \([0:10:150]\) determined the boundaries of the bins. As a result, the center of the bins will be given by

\[
\text{BinCenters} = [5:10:145];
\]

Before we continue, let's turn the filament lengths in pixels into actin monomers. First, we need to convert the bins to \(\mu\text{m}\) using Calibration

\[
\text{Lengthum} = \text{BinCenters} \times \text{Calibration}; \quad \%\text{length in \(\mu\text{m}\)}
\]

Finally, we make use of the fact that on actin monomer adds about 2.7 nm of length to the filament such that

\[
\text{LengthMonomer} = \text{Lengthum} / 2.7E-3; \quad \%\text{length in monomers}
\]

We can extract the number of filaments in each bin using HistHandle

\[
\text{NFilaments} = \text{HistHandle.Values};
\]

and the counts in each bins into a probability by dividing by the total number of filaments

\[
\text{ProbFilaments} = \text{NFilaments} / \text{sum(NFilaments)};
\]

We can now plot the histogram information as a scatter plot
Not surprinsigly, the first bin doesn't look like it would fit a line. This smaller bin includes all filaments that we cannot detect and hence it is to be expected for it to undercount the number of filaments. Also, longer filaments will be lost becasue of, for example, shearing due to pipetting. As a result, we will only trust the data in the [2:9] range. Let's fit this data to a line using polyfit

```matlab
Fit=polyfit(LengthMonomer(2:9),log(ProbFilaments(2:9)),1); %fit 1st degree polynomial (a line)
```

The resulting variable Fit has two elements corresponding to the slope and y-intercept such that

```matlab
plot(LengthMonomer,log(ProbFilaments),'.k')
xlabel('filament length (monomers)')
ylabel('probability')
hold on
plot(LengthMonomer,Fit(1)*LengthMonomer+Fit(2),'-r')
hold off
```
Not too bad! The data seems to confirm the prediction of our model. The distribution does indeed look exponential. However, we to remember that our model made a very specific quantitative prediction that goes much farther than the qualitative fact about the functional form of the probability distribution of filament lengths. Specifically, the model predicted that

\[ P(\ell) = \left( 1 - \frac{\ell}{\gamma} \right) e^{-\ell \log(\gamma/r)}. \]

If we take the ln of this distribution to match the way we plotted our data we get

\[ \ln(P(\ell)) = \ln \left( 1 - \frac{\ell}{\gamma} \right) - \ell \ln \left( \frac{\ell}{r} \right), \]

which tells us that the slope of our linear fit will be given by \(-\ln \left( \frac{\ell}{r} \right)\) and the y-intercept by \(\ln \left( 1 - \frac{\ell}{\gamma} \right)\). From the y-intercept, we can calculate the ratio \(\gamma/r\)

```matlab
gammaOverr = 1/(1-exp(Fit(2)))
gammaOverr = 1.5244
```

On the other hand, we can obtain this same ratio from the slope by just doing

```matlab
exp(-Fit(1))
ans = 1.0010
```
The slope obtained from the fit and the slope predicted by the y-intercept are within the same order of magnitude. Still, they differ by 50%! Note that $\gamma/r$ determines the decay length of the exponential distribution. As a result, it looks like the y-intercept demands a flatter distribution than that measured by fitting the slope directly. What are the potential reasons for such disagreement? Some possible ideas are

- The experiment did not reach steady-state in the 15 minutes of experiment. Therefore, it is not fair to compare non-steady-state data to a steady-state prediction such as that derived here.
- Other processes such as the known annealing of large filaments to each other describe the growth of longer filaments beyond the process of monomer incorporation modeled here.
- The assumption that the rate of monomer incorporation is constant is wrong because the monomer pool is being depleted.

While we won't explore these different hypotheses in detail, the reader is encouraged to do so or to propose other assumptions of our model that might have to be revisited. Regardless, it is clear that our analysis made it possible to go beyond the simple qualitative statement about the exponential shape of the distribution to extract quantitative data that makes it possible to more precisely and more critically put the assumptions behind our model to a stringent test.