# Southern Blots

April 8, 2015

## 1 Extracting quantitative information from the Southern blot data

To make any quantitative assessment of the telomere packing, we need to know the distribution of genomic lengths that the telomeres can have.

In principle, the Southern blot data can provide us information on this distribution. However, due to some technical considerations, the interpretation of the blots is not straight forward. In this worksheet I will develop the theory that lets us determine the mean and standard deviation of the genomic lengths distribution from the Southern blot data and apply it to the telomere blots.

```
In [1]: # Import libraries for numerics and plotting
        %pylab
        %matplotlib inline
```

```
Using matplotlib backend: TkAgg
Populating the interactive namespace from numpy and matplotlib
```

### 1.1 Theory of quantitative Southern blots

Let's consider how a Southern blot "signal" is generated. By signal I mean the (inverse) line profile taken through an image of the blot.

First, one has a probability density function that describes how long a given telomere is if one were to randomly pick a telomere out of a pool of telomeres. This density function is written as $p(L)$ with $L$ being the genomic length of the telomere. We want to know something about $p(L)$, but what we have is a Southern blot signal, which I denote by $s(L)$.

The primary complication comes from this fact: more probes can bind to longer telomeres, so longer telomeres will have more signal than shorter telomeres. An intuitive example would be to consider a population of two telomeres, one that is 1 kb long and one that is 5 kb long. The probability of picking either telomere is 1/2; they have equal probability. However, the signal coming from the 5 kb telomere will have $\frac{5}{l}$ times more signal, where $l$ is the average probe length, because more probes can bind to it. Thus, our Southern blot will report a higher value at 5 kb than at 1 kb.

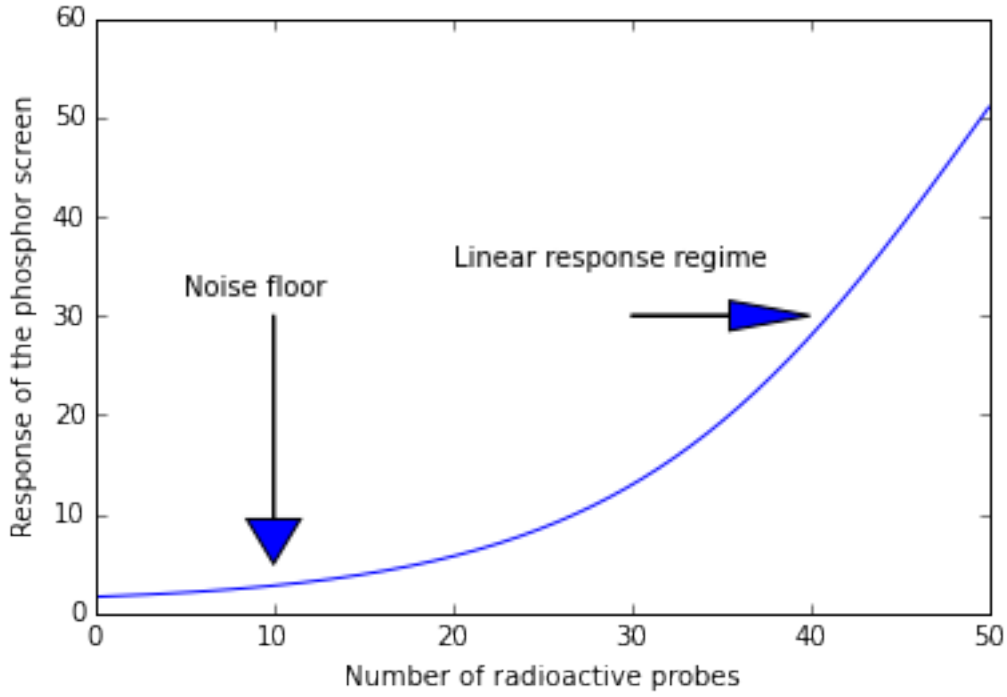
I expect that the total amount of radioactivity for one value of the genomic length $L$ should be proportional to the number of probes at the same value. The number of probes is the total number of telomeres in the population with that length, multiplied by the number of probes that can bind to a telomere at that length. The first quantity, the number of telomeres, is $N_{telomeres} \times p(L)$, where $N_{telomeres}$ is the total number of telomeres in my pool. The second quantity, the number of probes of length $l$ that can bind to a telomere of length $L$, is $\frac{L}{l}$. This means that the total number of probes for a given genomic length and population of telomeres is

$$N_{probes}(L) = N_{telomeres} \times p(L) \times \frac{L}{l} \tag{1}$$

The signal that we measure in the Southern is related to this quantity, but is not necessarily proportional to it. This is because the real signal has a lower limit (the noise floor) and an upper limit (the saturation point) which the signal cannot excede.

Since we haven't done any measurements on the saturation of the phosphor screen, we'll assume that the only nonlinearity is the noise floor. An illustration of this is below:

In [2]:
```
# Example is a logistic function that is linear at mid-range x values with a noise floor and sa
# (I cropped the large x-values since we don't know the saturation point for this illustration.
x = np.linspace(0, 50, 1e5)

# The noise floor is set to one here.
example = 1 + 1e2 / (1 + np.exp(-0.1*(x-50)))

plt.plot(x, example)
plt.xlabel('Number of radioactive probes')
plt.ylabel('Response of the phosphor screen')

plt.arrow(10, 30, 0, -25, head_width = 3, length_includes_head = True)
plt.annotate('Noise floor', (5, 32))

plt.arrow(30, 30, 10, 0, head_width = 3, length_includes_head = True)
plt.annotate('Linear response regime', (20, 35))
plt.show()
```




The Southern blot signal can be written as a composition of two functions: the first is the response of the phosphor screen, which I call $r\left(N_{probes}\right)$, and the second is the number of probes as a function of the genomic length described above, $N_{probes}\left(L\right) = N_{telomeres} \times p\left(L\right) \times \frac{L}{l}$. The Southern blot signal is thus

$$s\left(L\right) = \left(r \circ N_{probes}\right)\left(L\right) \tag{2}$$

**The goal is take measured the signal $s\left(L\right)$ and determine $p\left(L\right)$ from it.** Of course, this is a simplified model which ignores the impulse response of the blot and the smearing that's usually observed behind a blot in the track.

Because of the simplifications in the model, we'll only estimate the mean and standard deviation of $p(L)$, rather than the full shape of $p(L)$ itself. These are the only values necessary for the simulations, anyway.

## 1.2 Determining telomere length distributions

From the theory, we know that a line profile through the Southern blot images is related to the distribution of the telomere genomic lengths, weighted by a linear factor and subject to noise and plate saturation.

The goal is to take the Southern blot line profile, $s(L)$ and obtain the distribution of genomic lengths, $p(L)$. A complete analysis for doing this would have measured the noise floor and saturation point of the plate. Lacking this, we can only make assumptions as to their values.

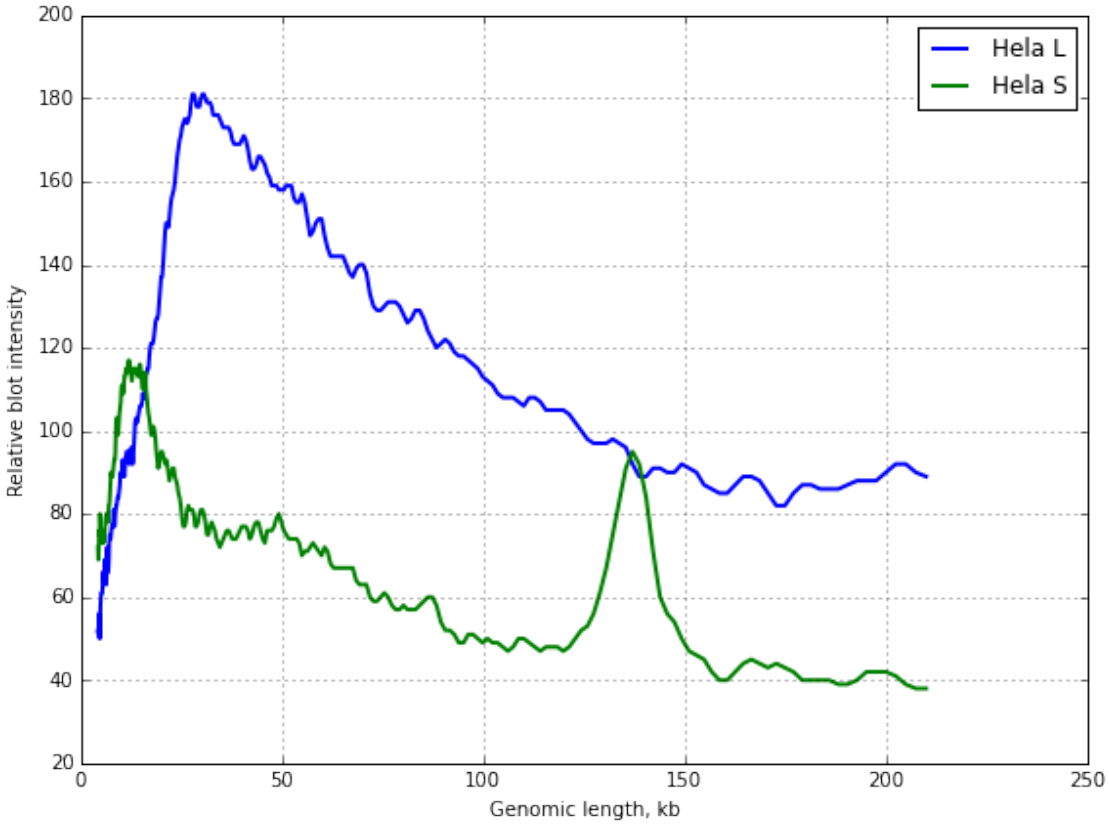
A simple assumption is that we are in the linear regime of the plate response and subject only to noise. In this case, the (inverse) line profile is proportional to the distribution and a weighting factor equivalent to $\frac{L}{l}$, all with a constant added that represents the noise floor. The weighting factor is there because longer telomeres will have proportionately more signal than shorter ones, a result of the fact that more oligonucleotides labels can bind to longer telomeres.

Mathmetically, under the assumption of linearity and noise, we write $s(L) \sim \left[p(L) \times \frac{L}{l}\right] + \text{noise}$, so that $p(L) \sim \frac{s(L)-\text{noise}}{L/l}$. Since we'll have to normalize by a constant, the probe length $l$ actually won't matter under these assumptions and we can write more simply

$$p(L) \sim \frac{s(L) - \text{noise}}{L} \tag{3}$$

**This means, to get the distribution of genomic lengths for our sample of telomeres, we need to subtract the noise floor from the Southern blot signals and then divide by the genomic length.**

```
In [3]: # Import (inverse) line profile data
        hl = np.loadtxt('HelaL_SBlot', delimiter = ',')
        hs = np.loadtxt('HelaS_SBlot', delimiter = ',')

        # Invert line profiles to look like distributions
        NL = hl[:,0]; BlotL = -hl[:,1] + 255
        NS = hs[:,0]; BlotS = -hs[:,1] + 255

        # Plot the line profiles
        fig = plt.figure(figsize = (8.0 , 6.0), dpi = 300)
        plt.plot(NL, BlotL,
                 label='Hela L',
                 linewidth = 2.0)
        plt.plot(NS, BlotS,
                 label='Hela S',
                 linewidth = 2.0,
                 linestyle = '-')

        fig.tight_layout(pad = 1.5)

        plt.xlabel('Genomic length, kb')
        plt.ylabel('Relative blot intensity')
        plt.grid(True)
        plt.legend()
        plt.show()
```




We can see from the plot that both distributions have a slight skew to the left (longer tails on the right). This could reflect either the weighting factor $L$ in the signal or simply undigested material in the blot. The peak in the Hela S blot was simply a speck of dirt in the Southern blot image.

The y-axis is simply a measure of the (inverse) intensity in the pixels in a line profile drawn through the blots.

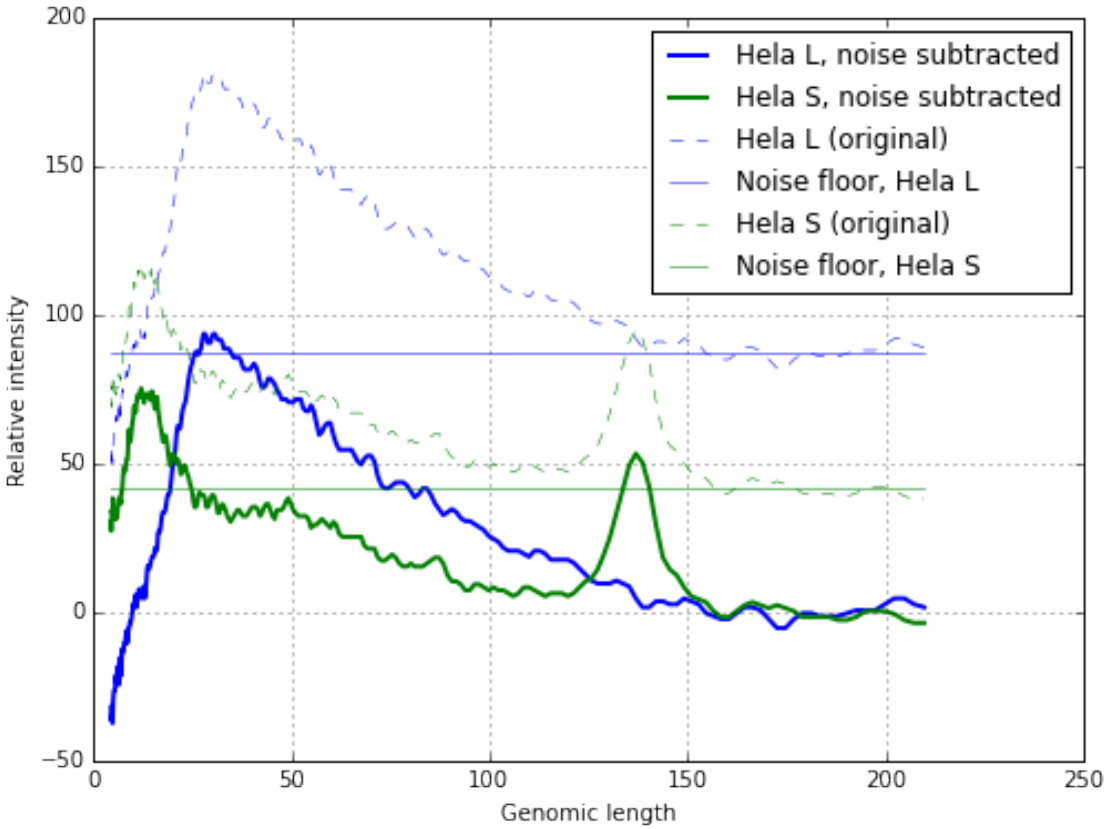
```
In [4]: # Peak locations in plots
        # (Multiple peaks may exist)
        print('Hela L peak(s): {0}'.format(NL[BlotL == max(BlotL)]))
        print('Hela S peak(s): {0}'.format(NS[BlotS == max(BlotS)]))
```

```
Hela L peak(s): [ 27.739  28.079  30.209  30.579]
Hela S peak(s): [ 11.964]
```

Based on the above, we might conclude that Hela L has a mean of about 28 kb and Hela S a mean of about 12 kb. However, remember that longer telomeres have proportionately more weight and there is a noise floor in these curves. To remove this effect (under the assumption of no saturation in the plate response), we need to first subtract the noise floor, and then divide these curves by $L$, the genomic length.

```
In [5]: # Noise is mean of plateaus in the line profile to the right of the peaks
        noiseL = np.mean(BlotL[NL > 150])
        noiseS = np.mean(BlotS[NS > 150])
        #noiseS = np.mean(BlotS[np.logical_and(NS > 25, NS < 50)])

        # Subtract noise floors
        bl_noNoise = BlotL - noiseL
```

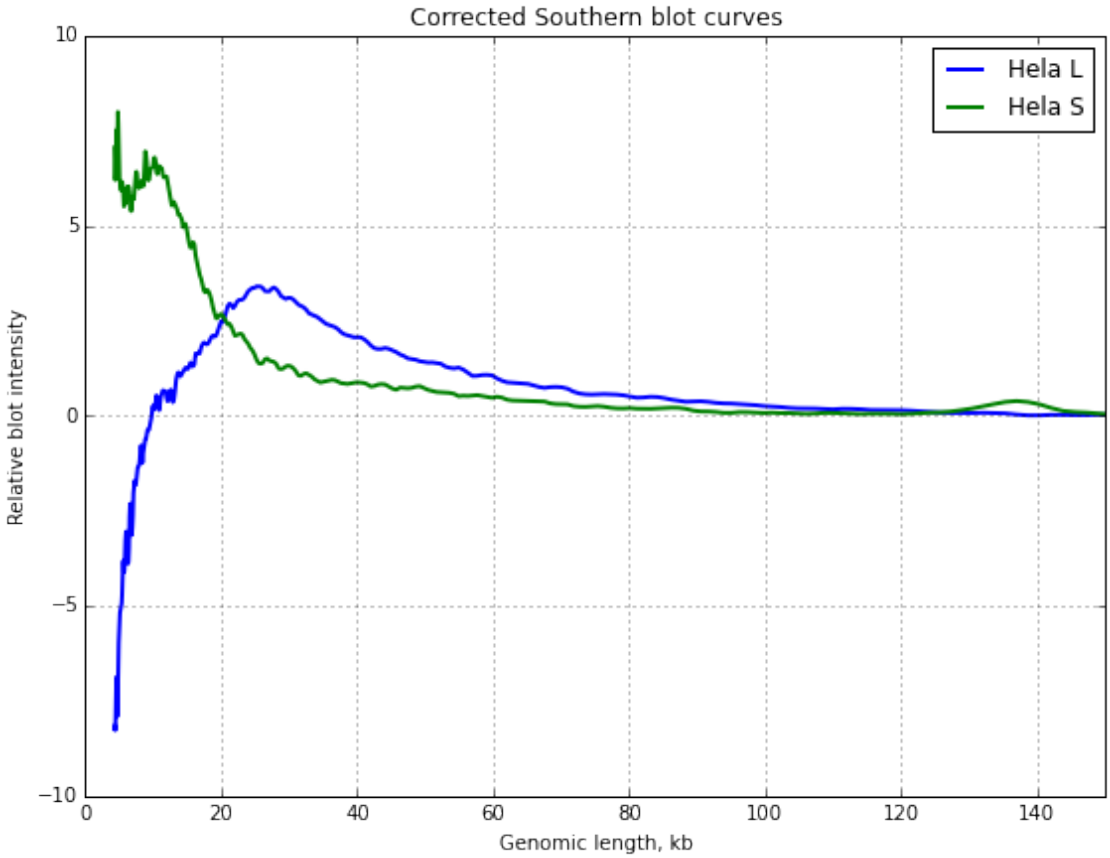
```
bs_noNoise = BlotS - noiseS

# Plot the original curves, the curves without noise, and the estimates of where the noise floo
fig = plt.figure(figsize = (8.0 , 6.0), dpi = 300)
plt.plot(NL, bl_noNoise,
         label='Hela L, noise subtracted',
         linewidth = 2.0)
plt.plot(NS, bs_noNoise,
         label='Hela S, noise subtracted',
         linewidth = 2.0,
         linestyle = '-')

plt.plot(NL, BlotL,
         label='Hela L (original)',
         linewidth = 0.5,
         linestyle = '--',
         color = 'blue')

plt.plot([NL[0], NL[-1]], [noiseL, noiseL],
         label='Noise floor, Hela L',
         linewidth = 0.5,
         linestyle = '-',
         color = 'blue')

plt.plot(NS, BlotS,
         label='Hela S (original)',
         linewidth = 0.5,
         linestyle = '--',
         color = 'green')

plt.plot([NS[0], NS[-1]], [noiseS, noiseS],
         label='Noise floor, Hela S',
         linewidth = 0.5,
         linestyle = '-',
         color = 'green')

plt.grid(True)
plt.legend()
plt.xlabel('Genomic length')
plt.ylabel('Relative intensity')
plt.show()
```




Now, we need to divide the new curves by the genomic length to remove the effects of longer telomeres having more probes.

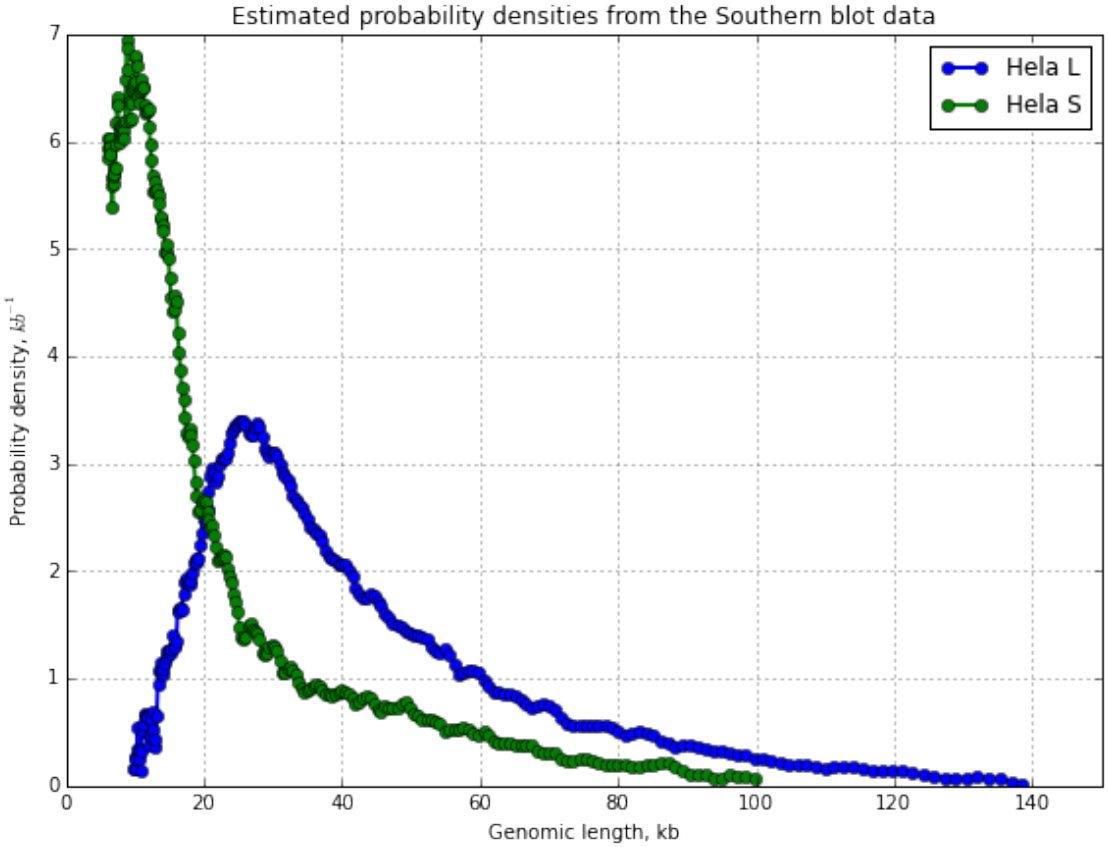
```
In [6]: # Correct curves for linear weighting by dividing the the genomic length
        correctedL = bl_noNoise / NL
        correctedS = bs_noNoise / NS

        # Plot the line profiles
        fig = plt.figure(figsize = (8.0 , 6.0), dpi = 300)
        plt.plot(NL, correctedL,
                 label='Hela L',
                 linewidth = 2.0)
        plt.plot(NS, correctedS,
                 label='Hela S',
                 linewidth = 2.0,
                 linestyle = '-')

        fig.tight_layout(pad = 1.5)

        plt.xlabel('Genomic length, kb')
        plt.ylabel('Relative blot intensity')
        plt.title('Corrected Southern blot curves')
        plt.xlim((0, 150))
        plt.ylim((-10, 10))
        plt.grid(True)
```

```
plt.legend()
plt.show()
```




In [7]:
```
# Crop out values less than zero (i.e., values very close to or less than the noise floor)
# Plot the probability estimates

NLRange = [9.8, 140]
NSRange = [6,   100]
#NSRange = [6, 25]

#NLRange = [16.3, 45]
#NSRange = [8, 19.5]

# Isolate the relevant peaks (first base pairs)
NpL = NL[np.logical_and(NL > NLRange[0], NL < NLRange[1])]
NpS = NS[np.logical_and(NS > NSRange[0], NS < NSRange[1])]

# The distributions
pL = correctedL[np.logical_and(NL > NLRange[0], NL < NLRange[1])]
pS = correctedS[np.logical_and(NS > NSRange[0], NS < NSRange[1])]

fig = plt.figure(figsize = (8.0 , 6.0), dpi = 300)
plt.plot(NpL, pL,
```

```
          label='Hela L',
          linewidth = 2.0,
          linestyle = '-',
          marker = 'o')
plt.plot(NpS, pS,
          label='Hela S',
          linewidth = 2.0,
          linestyle = '-',
          marker = 'o')

fig.tight_layout(pad = 1.5)

plt.xlabel('Genomic length, kb')
plt.ylabel(r'Probability density, $kb^{-1}$')
plt.title('Estimated probability densities from the Southern blot data')
plt.xlim((0, 150))
plt.grid(True)
plt.legend()
plt.show()
```




The above plot contains the **unnormalized** probability density functions for the Hela telomere genomic lengths. Keep in mind that the long tails are probably undigested material and extra labels that are usually seen as spurious smears in the blots.

### 1.2.1 Making numeric distributions from the Southern blot data

Now that we have the curves, let's turn them into distributions. We'll first convert the curves to histograms, and then normalize the histograms. This essentially creates probability mass functions.

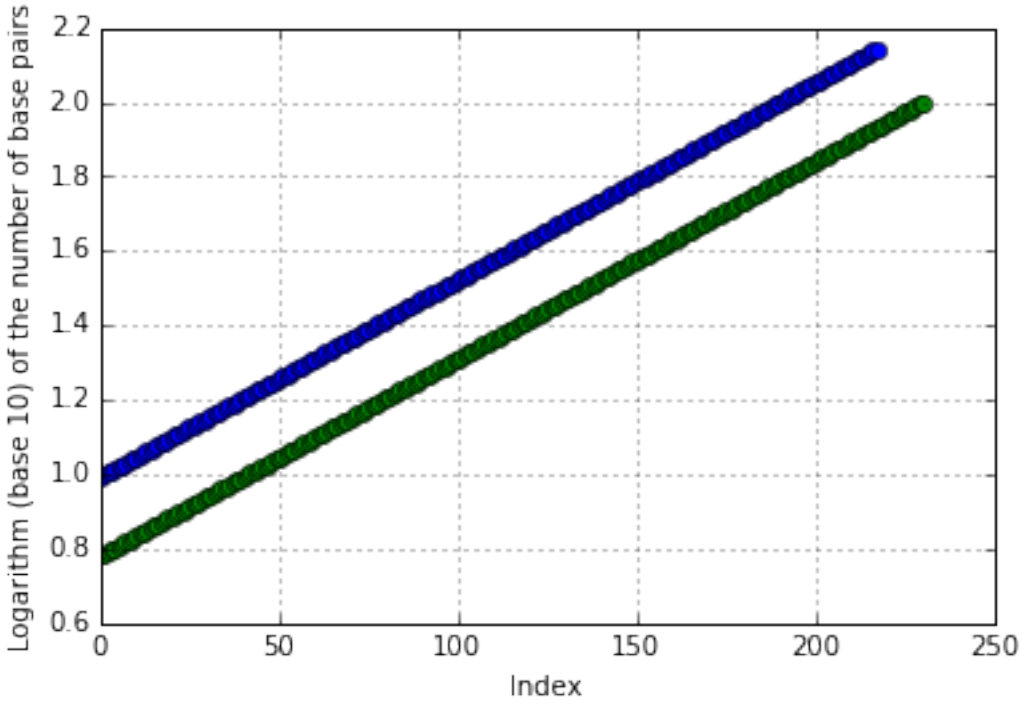
Note that even though the genomic length and label lengths are discrete variables, it should be ok to use continuous approximations up to this point. This is because the spacing between allowed values for the genomic length (1 base pair) and probe length (about 300 bp) is much smaller than the spread of the distributions (about 10,000 base pairs). The distributions thus essentially "look" continous.

First, and unfortunately, the numpy hist function accepts histogram bin *edges* as an argument. However, we have bin *centers*. We'll have to find the edges. This is done in a log10 scale since the data was originally obtained on this scale (blot ladders have logarithmic spacings).

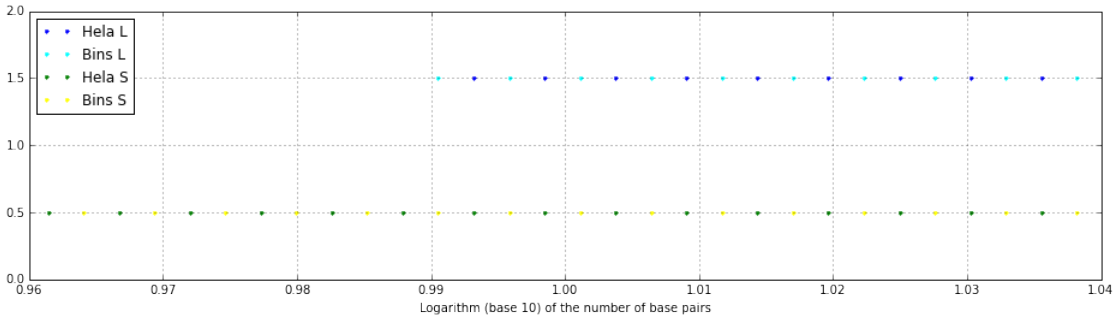
### 1.2.2 The following part on bin computations is not relevant to the discussion and can be skipped.

Please proceed to the section on normalizing the histograms below.

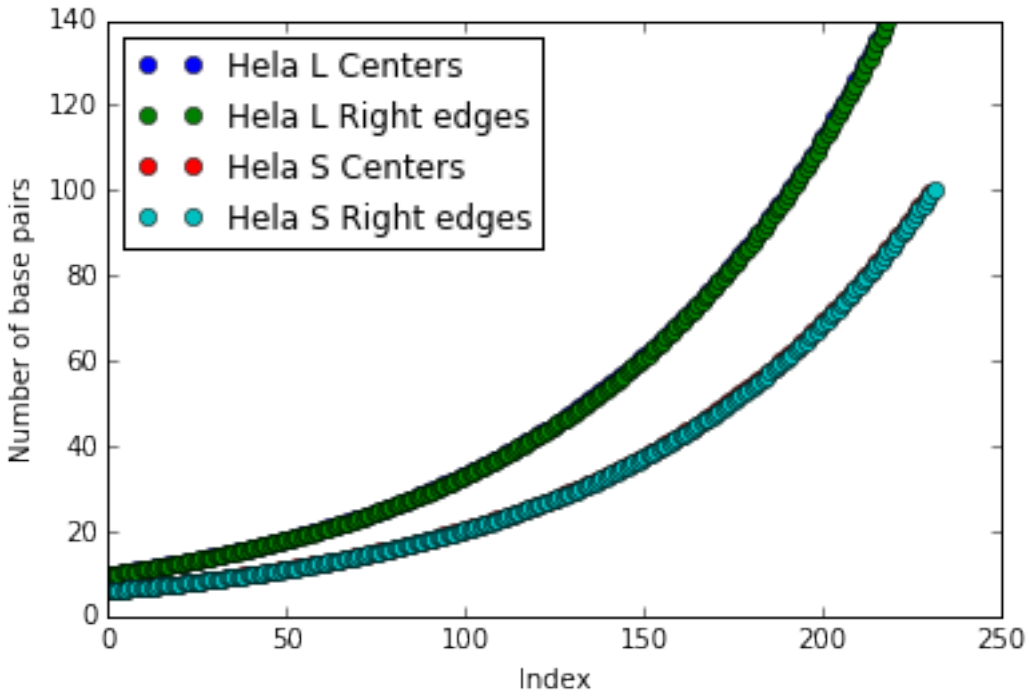
In [8]:
```
# Convert NpL and NpS to a log scale, which is the scale from which they were originally obtain
logNpL = np.log10(NpL)
logNpS = np.log10(NpS)

# Check that the scales are now linear
plt.plot(logNpL, 'o', label = 'Hela L')
plt.plot(logNpS, 'o', label = 'Hela S')
plt.xlabel('Index')
plt.ylabel('Logarithm (base 10) of the number of base pairs')
plt.grid()
```



In [9]:
```
# Spacing between adjacent points above
spacingL = logNpL[1] - logNpL[0]
spacingS = logNpS[1] - logNpS[0]

# Bin edges are the above points shifted to the left by half the spacing, with an additional po
logBinsL = logNpL - spacingL / 2
logBinsS = logNpS - spacingS / 2

logBinsL = np.append(logBinsL, logNpL[-1] + spacingL / 2)
logBinsS = np.append(logBinsS, logNpS[-1] + spacingS / 2)

# Check the spacings
yL = np.ones(len(logBinsL))  + 0.5
yS = np.ones(len(logBinsS))  - 0.5

plt.figure(figsize = (16, 4))
plt.plot(logNpL,    yL[0:-1], '.', label = 'Hela L', color = 'blue')
plt.plot(logBinsL,  yL,       '.', label = 'Bins L', color = 'cyan')
plt.plot(logNpS,    yS[0:-1], '.', label = 'Hela S', color = 'green')
plt.plot(logBinsS,  yS,       '.', label = 'Bins S', color = 'yellow')
plt.xlabel('Logarithm (base 10) of the number of base pairs')

plt.xlim(0.96, 1.04) # Zoom in to part of the plot so we can see the detail
plt.ylim(0,2)
plt.legend(loc = 'upper left')
plt.grid()
plt.show()
```

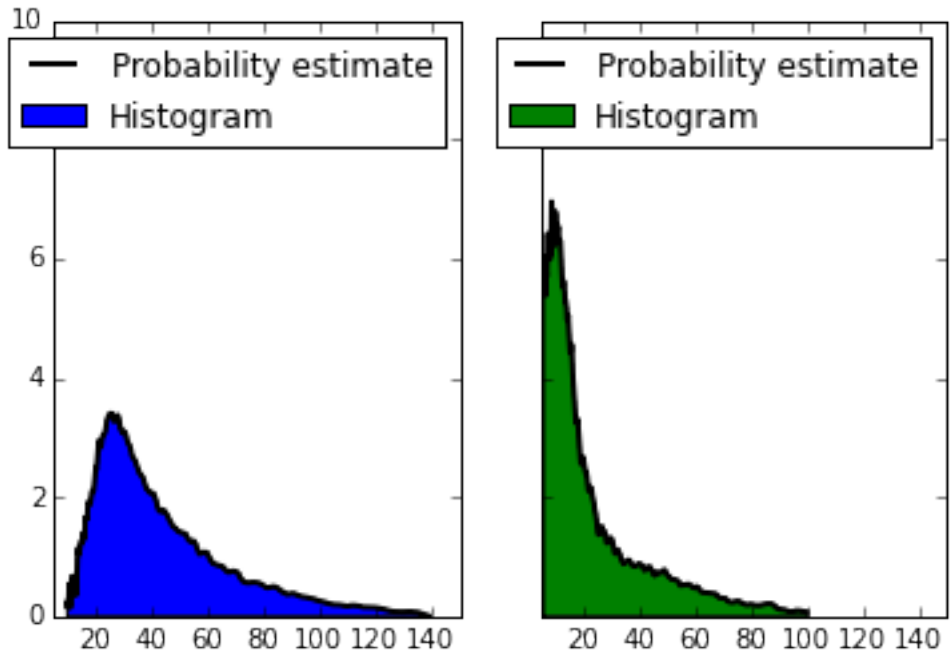



In [10]:
```
# Convert the bins back to a linear scale
binsL = 10**(logBinsL)
binsS = 10**(logBinsS)

# Check the conversion
plt.plot(NpL,   'o', label = 'Hela L Centers')
plt.plot(binsL, 'o', label = 'Hela L Right edges')
plt.plot(NpS,   'o', label = 'Hela S Centers')
plt.plot(binsS, 'o', label = 'Hela S Right edges')
plt.xlabel('Index')
plt.ylabel('Number of base pairs')
plt.legend(loc = 'upper left')
plt.show()
```




Now we can determine the histograms from the (non-normalized) probability densities above, and from the histograms get the normalized probability density estimates.

```
In [11]: histL, _ = np.histogram(NpL, bins = binsL, weights = pL, density = True)
         histS, _ = np.histogram(NpS, bins = binsS, weights = pS, density = True)

         # Check that the histograms and probabilities match
         fig, (ax1,ax2) = plt.subplots(1, 2, sharey = True)

         ax1.hist(NpL, bins = binsL, weights = pL, histtype = 'stepfilled', label = 'Histogram')
         ax1.plot(NpL, pL, 'k', linewidth = '2', label = 'Probability estimate')

         ax2.hist(NpS, bins = binsS, weights = pS, histtype = 'stepfilled', color = 'green', label = 'H
         ax2.plot(NpS, pS, 'k', linewidth = '2', label = 'Probability estimate')

         ax1.set_xlim((5, 150))
         ax2.set_xlim((5, 150))

         ax1.set_ylim((0, 10))

         ax1.legend()
         ax2.legend()
         plt.show()
```




Now, convert the histograms to probability mass densities.

### 1.2.3 Normalizing the histograms to get probability mass densities

By definition, the sum of the probabilities of the histogram bins times their width must be one. This can be achieved by dividing by a constant, a technique known as *normalization*. This is easily done in numpy by setting the *density* argument of np.histogram to **True**.

```
In [12]: # I don't need the bins output because I already have it
         histL, _ = np.histogram(NpL, bins = binsL, weights = pL, density = True)
         histS, _ = np.histogram(NpS, bins = binsS, weights = pS, density = True)

         # Check that the probability mass densities really are normalized (they should sum to 1 or ver
         diffL = np.diff(binsL)
         diffS = np.diff(binsS)

         print('Integral of Hela L probability mass density: {0}'.format(np.sum(histL * diffL)))
         print('Integral of Hela S probability mass density: {0}'.format(np.sum(histS * diffS)))
```

```
Integral of Hela L probability mass density: 1.0
Integral of Hela S probability mass density: 1.0
```

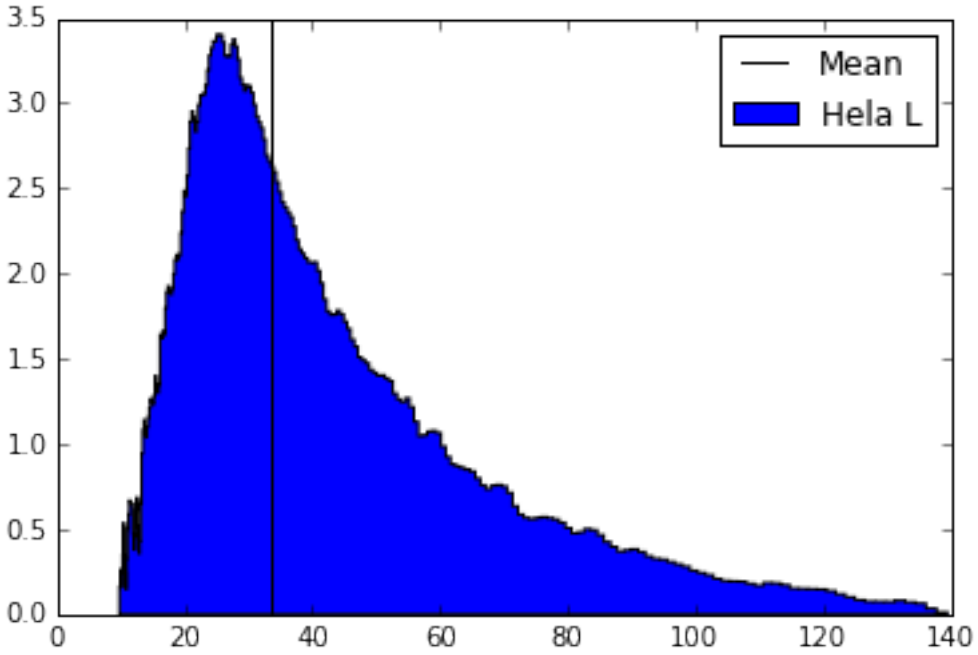
## 1.3 What are the mean and standard deviations of these distributions?

The mean of a probability mass density $p(L)$ is simply

$$\bar{L} = \sum_{i=0}^{N} L_i p(L_i) \Delta L$$

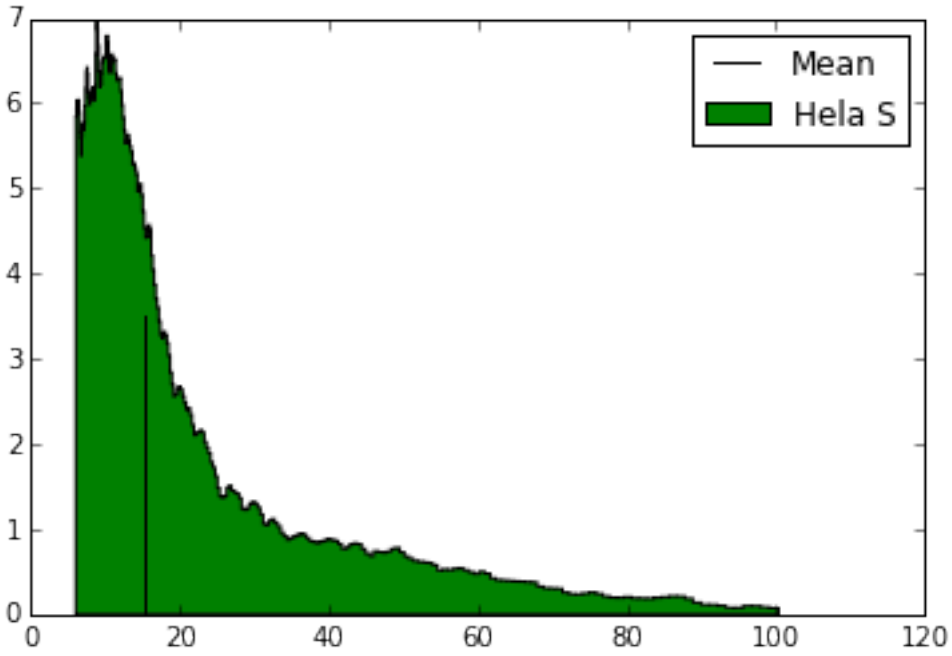
To compute this, I need to multiply the midpoint of each bin for the genomic length of the telomeres with its probability in the histL and histS arrays, and then sum them.

In [13]:
```
meanL = np.sum(NpL * histL * diffL)
meanS = np.sum(NpS * histS * diffS)

plt.hist(NpL, bins = binsL, weights = pL, histtype = 'stepfilled', label = 'Hela L')
plt.plot([meanL, meanL], [0, 3.5], 'k', label = 'Mean')
plt.legend()
plt.show()

print('Mean of Hela L: {0}'.format(meanL))
```




```
Mean of Hela L: 33.62511410242658
```

In [14]:
```
plt.hist(NpS, bins = binsS, weights = pS, histtype = 'stepfilled', color = 'green', label = 'H
plt.plot([meanS, meanS], [0, 3.5], 'k', label = 'Mean')
plt.legend()
plt.show()

print('Mean of Hela S: {0}'.format(meanS))
```




```
Mean of Hela S: 15.234458567853157
```

### 1.3.1 Estimate the standard deviations

```
In [15]: # Second central moments of the distributions
         secondCentMomL = np.sum((NpL - meanL)**2 * histL * diffL)
         secondCentMomS = np.sum((NpS - meanS)**2 * histS * diffS)

         # Standard deviations of the distributions
         stdL = np.sqrt(secondCentMomL)
         stdS = np.sqrt(secondCentMomS)

         print('Hela L standard deviation: {0:.2f}'.format(stdL))
         print('Hela S standard deviation: {0:.2f}'.format(stdS))
```

```
Hela L standard deviation: 18.89
Hela S standard deviation: 11.86
```

## 1.4 Maximum entropy distributions

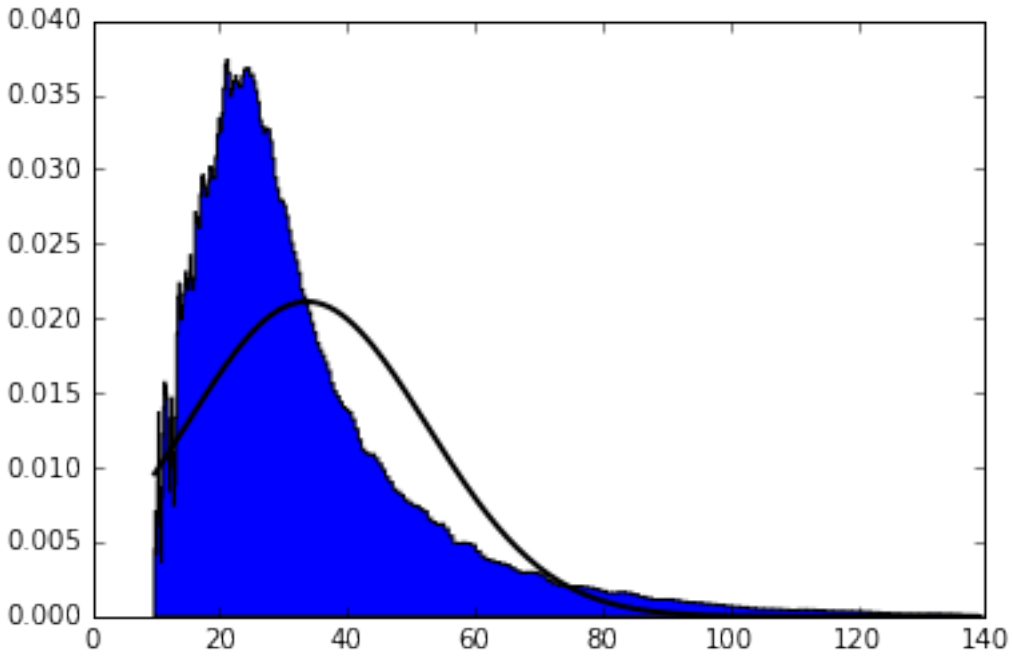
If all we know is the mean and standard deviation, maximum entropy states we should choose a Gaussian distribution for the probability. Here's what the equivalent Gaussian's look like relative to the measured probabilities.

```
In [16]: from scipy.stats import norm
```

```
In [17]: # Define Gaussians with the measured moments
         maxEntL = norm(loc = meanL, scale = stdL)
         maxEntS = norm(loc = meanS, scale = stdS)
```

In [18]:
```
plt.hist(NpL, bins = binsL, weights = pL, histtype = 'stepfilled', label = 'Hela L', normed = 
plt.plot(NpL, maxEntL.pdf(NpL), linewidth = 2, color = 'black')
plt.ylim((0, 0.04))
```

Out[18]: (0, 0.04)




The Gaussian with these parameters does not look very good. Why?

We also know, though, the the length can't be less than zero. This means that this maximum entropy model, which involves plugging in the mean and standard deviation into a Gaussian, is not likely correct.

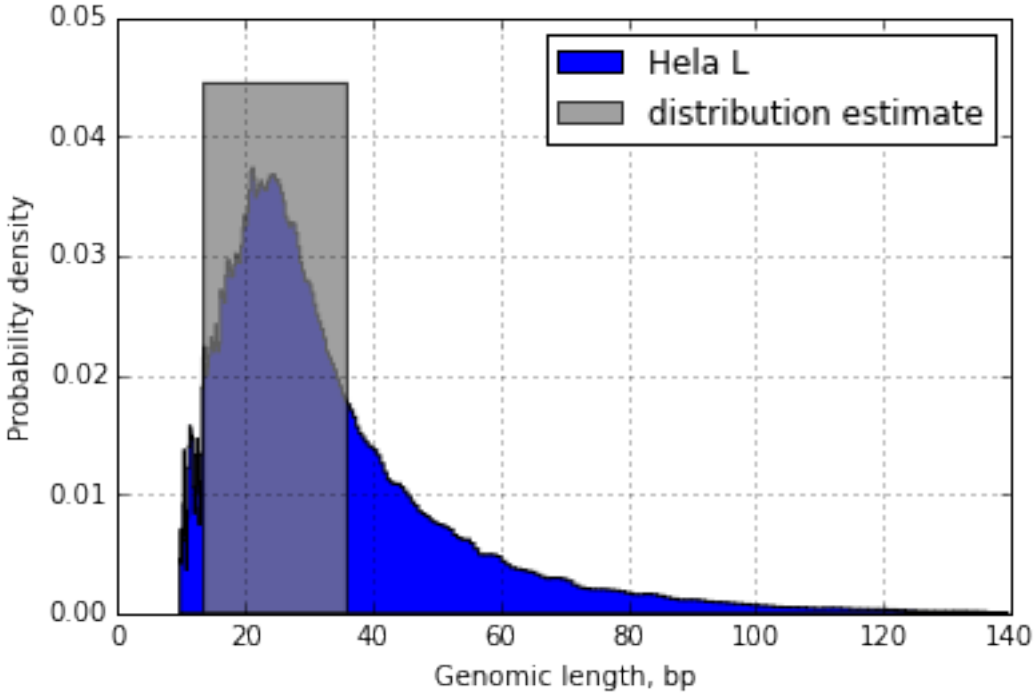
I also know the long tail at high genomic lengths is probably due to junk in the gel track (extra radio labels, undigested DNA, etc... I can see smears in the ladders), so the long tails should not be weighted so heavily. In fact, I would expect the larger the value of the distribution, the higher the weight we can place on the distribution. So really, we know quite a lot more than the mean and standard deviation.

In the end, it's probably best to do what I did before: use a uniform distribution with edges set to the full width half max of the corrected distributions above. **This step imposes the fewest assumptions on the distribution given what we know: the Southern blot data, the model to get the probability of genomic lengths, the fact that the genomic length can't be less than zero, and undigested material leads to a long tail in the measured distributions.**

Reassuringly, this gives almost the exact distributions as my previous estimates on a different blot, so this method is probably robust.

In [19]:
```
# Plot the histogram between the points correspoinding to half its maximum
plt.hist(NpL, bins = binsL, weights = pL, histtype = 'stepfilled', label = 'Hela L', normed = 

# Plot the approximate distribution to use for the simulations
plt.hist([24.7, 24.8], bins = [13.5, 24.75, 36], histtype = 'stepfilled', color = 'gray', alpha
plt.legend()
plt.xlabel('Genomic length, bp')
plt.ylabel('Probability density')
plt.ylim(0, 0.05)
```

```
plt.grid()
plt.show()
```




In [19]: