Genome Size Estimation Tutorial

**How can K-mer estimation help to find genome sizes?**

For a given sequence of length L,  and a k-mer size of k, the total k-mer’s possible will be given by **( L – k ) + 1**

 e.g. In a sequence of length of 14, and a k-mer length of 8, the number of k-mer’s generated will be:

**GATCCTACTGATGC** ( L = 14 ) on decomposition of k-mers of length k = 8,

Total number of k-mer's generated will be

n = ( L - k ) + 1

= (14 - 8 ) + 1

= 7

**GATCCTAC**, **ATCCTACT**,  **TCCTACTG**,  **CCTACTGA**, **CTACTGAT**, **TACTGATG**,  **ACTGATGC**

For shorter fragment sizes, as in the above example, the Total number of k-mers estimated is n = 7, and it is not close to actual fragment size of L which is 14 bps.  But for larger fragment size, the total number of k-mer’s (n) provide a good approximation to the actual genome size. The following table tries to illustrate the approximation:

|  |  |  |  |
| --- | --- | --- | --- |
| k=18 | |  |  |
| Genome Sizes | Total K-mers of k=18 | % error in genome estimation |  |
| L | N=(L-K)+1 |  |  |
| 100 | 83 | 17 |  |
| 1000 | 983 | 1.7 |  |
| 10000 | 9983 | 0.17 |  |
| 100000 | 99983 | 0.017 |  |
| 1000000 | 999983 | 0.0017 | **1MB genome size** |

So for a genome size of 1 Mb, the error between estimation and reality is only .0017%. Which is a very good approximation of actual size.

In the case of, 10 copies (C) of GATCCTACTGATGC sequence, then the total no of k-mer’s (n) of length k = 8  will be 70.

n = [( L - k ) + 1 ] \* C

= [(14 - 8 ) + 1] \* 10

= 70

To get the actual genome size, we simply need to divide the total by the number of copies:

= n / C

= 70 / 10

= 7

That will help us to understand, that we never sequence a single copy of genome but a population. Hence we end up sequencing C copies of genome. This is also referred as coverage in sequencing. To obtain actual genome size (N), divide the total k-mers (n) by coverage (C).

N = n / C

**k-mer Distribution of a Typical Real World Genome**

Major issue that is faced in a real world genome sequencing projects is a non-uniform coverage of genome. This can be accounted to technical and biological variables.

example:  biased amplification of certain genomic regions during PCR (a step in preparation of sequencing libraries) and presence of repetitive sequences in genome.

**k-mer size:**

The size of k-mers should be large enough allowing the k-mer to map uniquely to the genome (a concept used in designing primer/oligo length for PCR). Too large k-mers leads to overuse of computational resources.

In the first step, k-mer frequency is calculated to determine the coverage of genome achieved during sequencing. There are software tools like[Jellyfish](http://www.cbcb.umd.edu/software/jellyfish/) that helps in finding the k-mer frequency(the number of times) in sequencing projects. The k-mer frequency follows a pseudo-normal distribution (actually it is a Poisson distribution) around the mean coverage in histogram of k-mer counts.

Once the k-mer frequencies are calculated, a histogram is plotted to visualize the distribution and to calculate mean coverage.

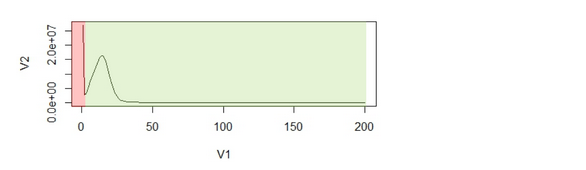
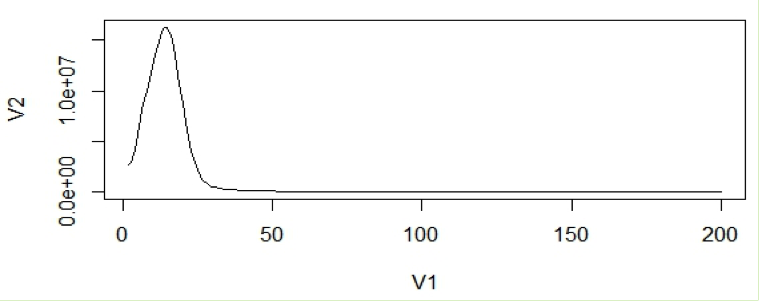


Figure 1: (A2) An example of k-mer histogram. The x-axis (V1), is the frequency or the number of times a given k-mer is observed in the sequencing data. The y-axis (V2), is the total number of k-mers with a given frequency.

The first peak is (in red region) primarily due to rare and random sequencing errors in reads. The values in graph can be trimmed to remove reads with sequencing errors.



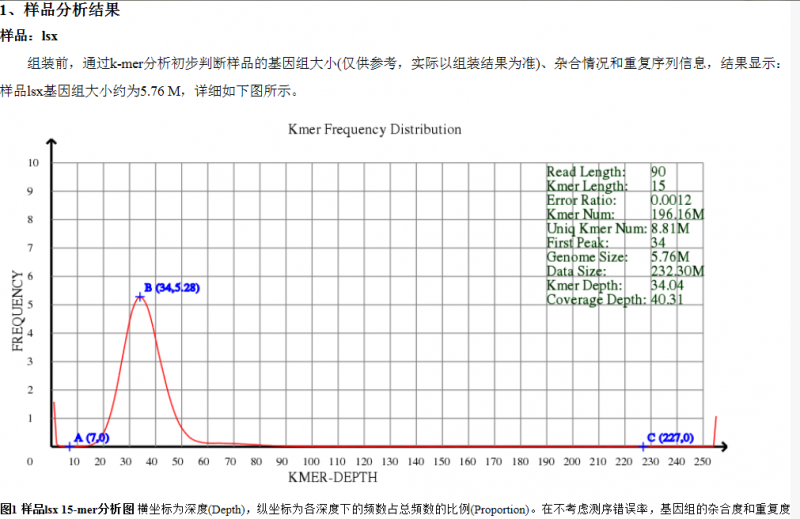
**With the assumption that k-mers are uniquely mapped to genome, they should be present only once in a genome sequence. So their frequency will reflect the coverage of the genome.**

For calculation purposes we use mean coverage which is 14 in above graph. The area under the curve will represent the total number of k-mers.

So the genome estimation will be:

N = Total no. of k-mers/Coverage

= Area under curve /mean coverage(14)



**Maximizing Utility of Available RAMs in K-mer World**

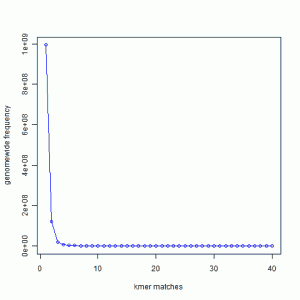
Bioinformaticians trying to assemble genomes or transcriptomes from large NGS libraries usually grapple with two problems – (i) how to set parameters to get the best assembly, and (ii) how to complete the assembly within [RAM](https://www.plob.org/tag/ram/) limits of the computer. Reaching limits of available [RAM](https://www.plob.org/tag/ram/) is always a challenge with large sequence libraries, no matter how powerful our computers are. The simplest solutions are either to remove a random subset of [reads](https://www.plob.org/tag/reads/) or trim [reads](https://www.plob.org/tag/reads/) from their 3′ ends on an ad hoc basis. Are they optimal? More sophisticated versions of above solutions are to remove duplicate [reads](https://www.plob.org/tag/reads/) instead of random [reads](https://www.plob.org/tag/reads/), or to use quality scores for trimming ends of the [reads](https://www.plob.org/tag/reads/). Are multiple copies of a read really redundant to the de Bruijn assemblers? We believe the researchers need to reorient themselves to k-mer world view to design most optimal cleaning strategies.

Let us say you intend to do a genome assembly from 200 million reads in a computer with 128 GB [RAM](https://www.plob.org/tag/ram/). The amount of [RAM](https://www.plob.org/tag/ram/) is insufficient for your computer, and therefore you need to cut down on the input library size. Conceptually everyone understands that keeping the best reads and removing the worst ones would produce the best result. It is just that the reads do not come marked with good or bad except for the quality score from the sequencers, but the quality scores are not always helpful for small errors.

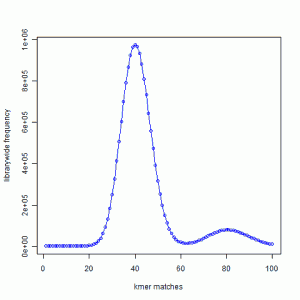
Let us handle the problem in a different way. We shall plot the distribution of k-mers in the data, and show that the distribution has two parts – signal and noise. The goal of best cleaning strategy is to reduce noise and keep signal Why do we check the k-mers? Because that is how de Bruijn assemblers view the sequence data.

If you remember our earlier discussion on de Bruijn graphs, we [first constructed de Bruijn graph of a genome](https://www.plob.org/2012/06/06/2067.html) and [then showed](https://www.plob.org/2012/06/06/2082.html) that de Bruijn graph constructed from short reads is nearly identical to the de Bruijn graph of the underlying genome. In this discussion, we will follow similar approach, but instead of going all the way to de Bruijn graphs, we shall restrict ourselves to the K-mers only.

In the following chart, we present the 21-mer distribution of sea urchin (*Strongylocentrotus purpuratus*) genome. The genome is approximately 900 Mbase long. To prepare this chart, we first split both strands of the genome into all possible 21-mers, and then collected the 21-mers into groups. Most 21-mers were unique in the genome, i.e. they were present only once. Those are 21-mers from non-repetitive parts of the genome. **To give you an exact number, 994,401,729 21-mers were present only once in the genome. That is the number you see for x=1 in the following graph. 122,161,264 21-mers were present twice (x=2).** 20,077,544 21-mers were present thrice (x=3). So, x axis of the following chart shows multiplicity of 21-mers and y axis shows total count for that multiplicity in the sea urchin genome. At the other end of the curve, 10,000 21-mers were present 67 times in the genome. Those were the repeat regions.

[](https://static.plob.ybzhao.com/wp-content/uploads/2012/06/14.png)

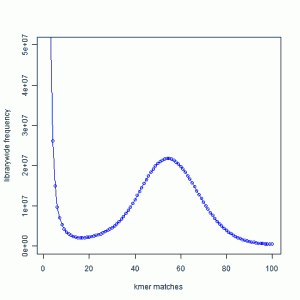
In the next exercise, we created simulated short reads (100-mer) by sampling the genome uniformly. The reads had 50X coverage on the genome, but all reads were error-free. Here is the K-mer distribution of the short reads generated in the same way as the above plot. Please note that Y-axis in the following plot is scaled down.

[](https://static.plob.ybzhao.com/wp-content/uploads/2012/06/26.png)

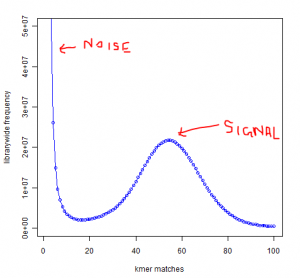
We see a large peak at x=40 instead of x=1. What does it mean? By way of construction, every part of the genome is sampled nearly 50 times. A subset of those short reads (about 20%) fell on repetitive regions. Therefore, the unique regions of the genome were sampled 40 times. **So, every K-mer that was present once in the genome is now present about 40 times in the short read library.**

**You also see a smaller peak at 80. That comes from those K-mers that were present 2 times in the genome. In fact, we checked and found out that the ratio of the peak size at 80 and peak size at 40 in the simulated read data is equal to the ratio of peak size at 2 and peak size at 1 in the K-mer distribution of the genome.**

In the next figure, we show the K-mer distribution constructed from short read library of another genome that we are trying to assemble. This is real data from 100 nucleotide Solexa libraries (HiSEQ), not simulated data.

[](https://static.plob.ybzhao.com/wp-content/uploads/2012/06/34.png)

You see a peak at around 55 suggesting that the short reads sampled the unique parts of the genome 55 times as expected from simulated data. However, you also see a large peak near 1. That is noise.

[](https://static.plob.ybzhao.com/wp-content/uploads/2012/06/42.png)

Why is the noise peak so big? Think about this - every time a read has one nucleotide error, it can result in 21 erroneous 21-mers. Thus, when we look at the k-mer distribution of all reads, errors appear magnified k times. However, there is also memory cost to having those errors, because Each of those erroneous k-mer occupies an unique space in the computer memory. Therefore, much of the RAM is actually wasted by the erroneous k-mers rather than real k-mers.

Clearly, a good strategy for cleaning data would be to reduce the noise peak as much as possible. When we trim 3′ edges, it makes sense to check what trimming is doing to the noise peak. If cutting one additional nucleotide does not reduce amount of noise, it is not helping the assembly. I think checking the k-mer distribution is a better measure than using the pre-assigned quality scores from the sequencer.

Removing duplicate reads reduces the size of signal peak compared to noise peak, and therefore it is a bad strategy. If one is keen on removing fraction of reads, he should try to find the reads that give rise to faulty k-mers and remove them. That step will also result in far better memory usage than removing duplicate reads.

Article From：

http://www.homolog.us/blogs/2011/09/20/maximizing-utility-of-available-rams-in-k-mer-world/

**Explain!!!**

<http://seqanswers.com/forums/showthread.php?t=10988>

The BGI method is based around the observation that the coverage achieved for a genome is based on the size of the genome and the total amount of sequence data generated. So if you sequence 100 Mb of data for a 10 Mb genome, you should get ~10-fold coverage.   
Or as a simple equation: depth of coverage = total data / genome length.  
If you have any two of these parameters (i.e., you know the amount of data you generated and you know the genome size) obviously you can calculate the third.  
Usually when doing de novo genome sequencing you don't know the genome size, and since you don't have the genome, you don't know the coverage, but you do know how much data you've generated (i.e., the 'total sequence length' to use BGI's term). To estimate the genome size, you then need to estimate the coverage depth (N).   
To do this, you can calculate the kmer frequency within your read data (most people will do this for one of their small insert libraries for which they have the most information). Meaning you chop all of the reads you've generated up in to kmers (a kmer of 17 is the most common, as it is long enough to yield fairly specific sequences (meaning that it is unlikely the kmer is repeated throughout the genome by chance), but short enough to give you lots of data). **You then count the frequency with which each 17-mer represented by your data is found among all of the reads generated and create a frequency histogram of this information.** For non-repetitive regions of the genome, this histogram should be normally distributed around a single peak (although in real data you will have a asymptote near 1 because of rare sequencing errors etc). **This peak value (or peak depth) is the mean kmer coverage for your data.**   
You can relate this value to the actual coverage of your genome using the formula **M = N \* (L – K + 1) / L**, where M is the mean kmer coverage, N is the actual coverage of the genome, L is the mean read length and k is the kmer size.  
L -k +1 gives you the number of kmers created per read.   
  
So basically what the formula says is the kmer coverage for a genome is equal to the mean read coverage \* the number of kmers per read divided by the read length.  
Because you know L (your mean read length) and k (the kmer you used to estimate peak kmer coverage) and you've calculated M (soapdenovo comes with a script called kmerfreq that will this), you simply solve the equation for N as:  
N = M/((L-k+1)/L) and calculate N.  
Once you have that, divide your total sequence data by N and you have your genome estimate.

**Methodology**

The study was conducted to estimate the genome size of the species with low coverage short read data to validate existing estimates (flow cytometry sourced) or produce a new computational estimate. The genome size is calculated from short sub-sequences (k-mers) of Illumina short read data. A larger k-mer size need to be considered for the genome estimation.

Reads should be quality controlled before the genome estimate is provided. Numerous programs exist for this purpose but [Sickle](https://github.com/najoshi/sickle)(https://github.com/ucdavis-bioinformatics/sickle) is the application of choice in this tutorial. We require a minimum Phred-scaled quality value of 25 to for estimates.

Upon performing quality control, the k-mer distribution was calculated using the [Jellyfish](http://www.cbcb.umd.edu/software/jellyfish/) k-mer counting program. Then a histogram was constructed to perform the genome estimation using the same program.

The R statistical package is used to plot the binned distributions for the selected k-mer lengths. Initally the full data set is plotted and initial data points are often very high number due to the low frequency data points, and thus it should be avoided. Once the peak position is determined the total number of k-mers in the distribution is calculated, and then the genome size can be estimated using the peak position. In the ideal situation (or theoretically) the peak shape should be a Poisson distribution. In order to come up with a k-mer size a number of k-mer sizes are selected and genome estimation is done, where we see a regular distribution of the genome size.

Data sets used in this tutorial are available on BBC cluster:

Path: /common/Tutorial/Genome\_estimation

Data sets: sample\_read\_1.fastq   sample\_read\_2.fastq

Script: GenomeEstimationScript.sh

**Tutorial Outline**

1. [Count k-mer occurrence using Jellyfish 2.2.6](https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#outline1)
2. [Generate histogram using R](https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#outline2)
3. [Determine the single copy region and the total k-mers](https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#outline3)
4. [Determine peak position and genome size](https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#outline4)
5. [Compare the peak shape with Poisson distribution](https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#outline5)

**1. Count k-mer occurrence using Jellyfish 2.2.6**

jellyfish count -t 8 -C -m 19 -s 5G -o 19mer\_out --min-qual-char=? /common/Tutorial/Genome\_estimation/sample\_read\_1.fastq /common/Tutorial/Genome\_estimation/sample\_read\_2.fastq

options used in the counting k-mers:

-t -treads=unit32       Number of treads to be used in the run. eg: 1,2,3,..etc.

-C -both-strands        Count both strands

-m -mer-len=unit32     Length of the k-mer

-s -size=unit32         Hash size / memory allocation

-o -output=string       Output file name

--min-quality-char         Base quality value. Version 2.2.3 of Jellyfish uses the “Phred” score, where "?" = 30

If you need to look at a detailed usage of ‘count’ in jellyfish, type the following after loading the jellyfish module.

jellyfish <cmd> [options]

jellyfish count --help

or

jellyfish count --full-help

In order to run the program Jellyfish in the bbc cluster, compose the following script and save it or you can copy GenomeEstimationScript.sh from the bbc cluster, which is located at the following [path](https://bioinformatics.uconn.edu/genome-size-estimation-tutorial/#tutorial_path). (More on the commands used in the script can be found in [~~here~~](http://bioinformatics.uconn.edu/understanding-the-bbc-cluster-and-sge/))

#!/bin/bash

#$ -N jellyfish

#$ -M user\_email\_id

#$ -q highmem.q

#$ -m bea

#$ -S /bin/bash

#$ -cwd

#$ -pe smp 8

#$ -o JellyFish\_$JOB\_ID.out

#$ -e Jellyfish\_$JOB\_ID.err

module load jellyfish/2.2.6

jellyfish count -t 8 -C -m 19 -s 5G -o 19mer\_out --min-qual-char=? /common/Tutorial/Genome\_estimation/sample\_read\_1.fastq /common/Tutorial/Genome\_estimation/sample\_read\_2.fastq

It will create a out put file called 19mer\_out. Then using the above created file (19mer\_out), data points for a histogram will be created using the following command:

jellyfish histo -o 19mer\_out.histo 19mer\_out

If you open the 19mer\_out.histo file, it will have the frequency counting of each k-mer with the k-mer length of 19.

1 13016694

2 3159677

3 3938273

4 5416130

5 7140173

6 8860956

7 10461902

8 11938782

9 13277372

10 14419501

11 15230762

12 15594907

13 15416499

14 14655690

15 13442589

16 11834646

17 10049240

18 8239408

19 6533641

20 5047445

21 3808930

22 2817665

23 2069050

24 1522365

25 1134562

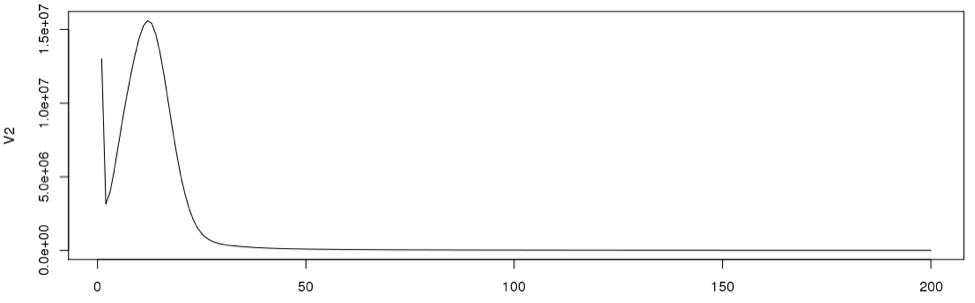
**2. Plot results using R**

To visualize and to plot the data we use R Studio. Using the following command, we will load the data from the out\_19mer.histo file in to a data frame called dataframe19.

dataframe19 <- read.table("19mer\_out.histo") #load the data into dataframe19

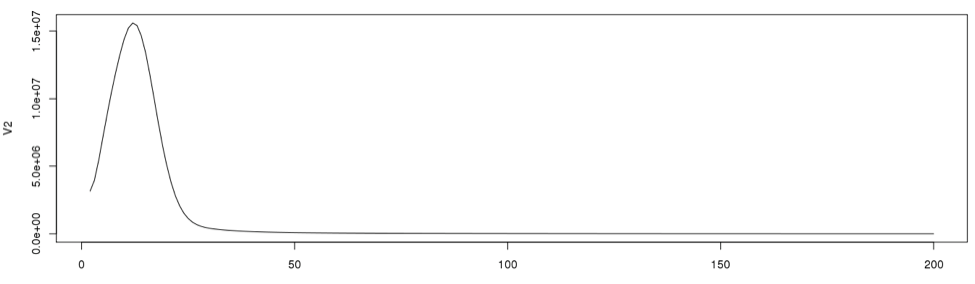
plot(dataframe19[1:200,], type="l") #plots the data points 1 through 200 in the dataframe19 using a line

This will create a line graph using the first 200 data points and the graph will look like the following:



In general, very low frequency k-mers represent high numbers that would skew the y –axis. If we look at the data points in the histogram file, we can see that the very first data point has a exceptionally high value than the next (second) data point. So we remove just the first line and re-plot using R. From now on we will disregard the first data point for our calculations.

plot(dataframe19[2:200,], type="l")



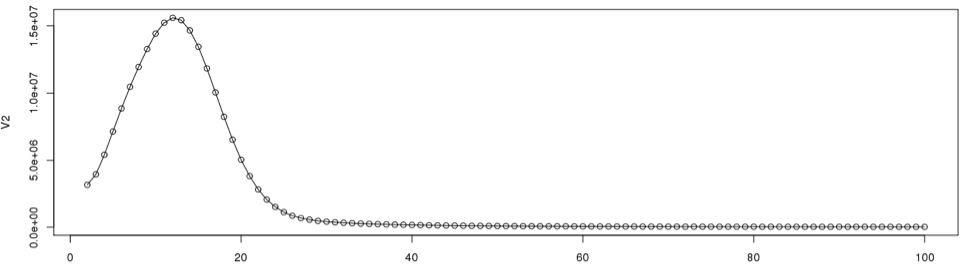
**3. Determine the single copy region and the total number of k-mers**

To determine the cut off points in the single copy region, we need to see the actual data point positions in the graph. In the initial examination the peak starts from the 2nd data point onward. So we will disregard the first data point in determining the single copy region.

Now using R we will re-plot the graph to determine the single copy region. Then we will include the points for clarity on the same graph.

plot(dataframe19[2:100,], type="l") #plot line graph

points(dataframe19[2:100,]) #plot the data points from 2 through 100



According to the graph, the single copy region would be between points 2 and 28.

Assuming the total number of data points is 9325, we can now calculate the total k-mers in the distribution.

sum(as.numeric(dataframe19[2:9325,1]\*dataframe19[2:9325,2]))

It will produce the results as:

[1] 3667909811

**4. Determine peak position and genome size**

From the plotted graph we can get an idea where the peak position lies, and it should be between 5-20 data points. Now to determine the peak, thus, we need to look at the actual data points in that region. Using the below command we will examine the actual data points between 5 and 20.

data[10:20,]

   V1   V2

5   5 7140173

6   6 8860956

7   7 10461902

8   8 11938782

9   9 13277372

10 10 14419501

11 11 15230762

12 12 15594907

13 13 15416499

14 14 14655690

15 15 13442589

16 16 11834646

17 17 10049240

18 18 8239408

19 19 6533641

In this case the peak is at, 12. So the Genome Size can be estimated as:

sum(as.numeric(dataframe19[2:9325,1]\*dataframe19[2:9325,2]))/12

Where the genome size is:

[1] 305659151

~ 305 Mb

It would be interesting to see the proportionality of the single copy region compared to the total genome size. In this data set the single copy region is between data point 2 and 28, So the size of single copy region can be roughly calculated as:

sum(as.numeric(dataframe19[2:28,1]\*dataframe19[2:28,2]))/12

[1] 213956126

~ 213 Mb

The proportion can be calculated as:

(sum(as.numeric(dataframe19[2:28,1]\*dataframe19[2:28,2]))) / (sum(as.numeric(dataframe19[2:9325,1]\*dataframe19[2:9325,2])))

[1] 0.6999827

~ 70 %

**5. Compare the peak shape with Poisson distribution**

Since we have a nice curve, we can compare our curve to a theoretical curve, which is the Poission distribution.

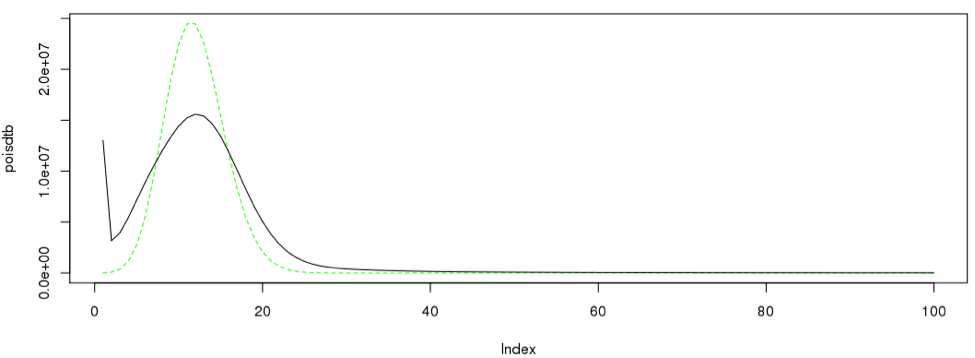
singleC <- sum(as.numeric(dataframe19[2:28,1]\*dataframe19[2:28,2]))/12

poisdtb <- dpois(1:100,12)\*singleC

plot(poisdtb, type='l', lty=2, col="green")

lines(dataframe19[1:100,12] \* singleC, type = "l", col=3)#, Ity=2)

lines(dataframe19[1:100,],type= "l")



Likewise the procedure can be iterated across different number of k-mers.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| K-mer length | 19 | 21 | 23 | 25 | 27 | 29 | 31 |
| total No of fields | 9741 | 9653 | 9436 | 9325 | 9160 | 8939 | 8818 |
| Total K-mer count | 3.66E+09 | 4.4E+09 | 4.53E+09 | 4.67E+09 | 4.26E+09 | 4.44E+09 | 3.98E+09 |
| Genome size | 3.05E+08 | 2.9E+08 | 3.02E+08 | 3.14E+08 | 3.04E+08 | 3.41E+08 | 3.06E+08 |
| single copy region | 2.13E+08 | 2E+08 | 2.08E+08 | 2.25E+08 | 2.21E+08 | 2.36E+08 | 2.3E+08 |
| Proportion | 0.69998 | 0.69403 | 0.688915 | 0.716151 | 0.725814 | 0.690277 | 0.750801 |