Load libraries

In [3]:
```
import numpy as np                                          #NumPy
from os import listdir                                      #List items in directory
from Bio.Align import PairwiseAligner                       #Used to align sequences (my 1
import matplotlib.pyplot as plt                             #Data visualization (for the p
from scipy.cluster.hierarchy import dendrogram              #Plot a dendrogram/phylogeneti
from scipy.cluster.hierarchy import linkage                 #Clustering required to by den
from matplotlib.pyplot import figure                        #Used to adjust the figure siz
from Bio import AlignIO                                     #Used to import an aligned fas
```

Create a FASTA node

In [4]:
```
class FASTA():
    def __init__(self, name, sequence, animal):
        self.name = name
        self.sequence = sequence
        self.animal = animal
```

Load FASTA files while also combining the data into 1 file

Soy Leghemoglobin Data

In [5]:
```
leg_data = []    # Basic data set
combined = open("Alignment\leg_data_unaligned.fasta", "w")

for filename in listdir("Raw Data\Soy Leghemoglobin"):
    file = open("Raw Data\Soy Leghemoglobin\\"+filename, "r")
    name = file.readline()[1:]
    combined.write(">"+name)

    sequence = ""
    for line in file.read().split("\n"):
        if not line == "":
            sequence += line
    combined.write(sequence+"\n\n")

    leg_data.append(FASTA(name, sequence, "Soy"))

    file.close()
combined.close()
```

Various Myoglobin Data

In [6]:
```
myo_data = []
combined = open("Alignment\myo_data_unaligned.fasta", "w")

for filename in listdir("Raw Data\Various Myoglobin"):
    file = open("Raw Data\Various Myoglobin\\"+filename, "r")
    name = file.readline()[1:]
    combined.write(">"+name)

    sequence = ""
    for line in file.read().split("\n"):
        if not line == "":
            sequence += line
    combined.write(sequence+"\n\n")
    myo_data.append(FASTA(name, sequence, filename[:filename.index(" ")]))

    file.close()
combined.close()
```

Build a Phylogenetic Tree using a Distance Matrix

In [7]:

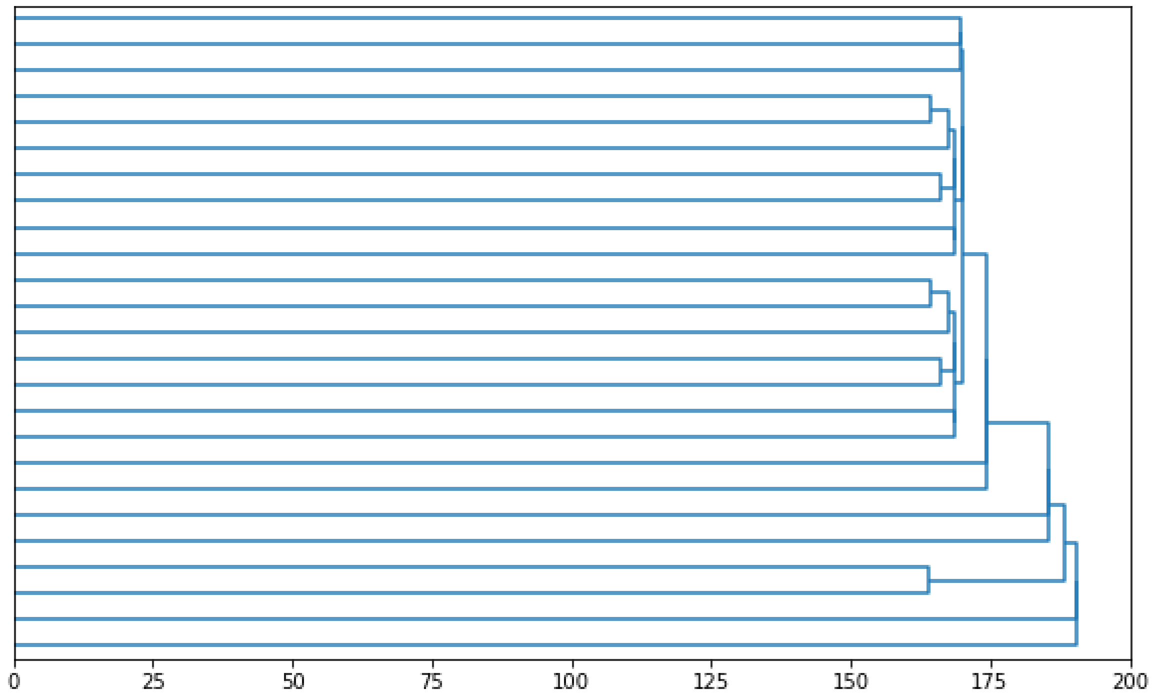
```
def build_tree(data, labels = None):
    # Create a distance matrix from the sequences
    dists = []
    for i in range(len(data)-1):
        dists_i = []
        for j in range(i+1, len(data)):
            alignment = PairwiseAligner().align(data[i].sequence,data[j].sequence)
            dists_i.append(alignment.score)
        dists.append(dists_i)
    # Fill repetitive half of distance matrix
    dists.append([])
    dists = np.flip(dists).tolist()
    for i in range(int((len(dists)+1)/2)):
        dist_save = dists[i]
        dists[i] = dists[i] + [0] + np.flip(dists[len(dists)-i-1]).tolist()
        if i!=len(dists)-i-1:
            dists[len(dists)-i-1] = dists[len(dists)-i-1] + [0] + np.flip(dist_save).to
    # Build tree
    linkage_matrix = linkage(dists)
    no_labels = False
    if labels == None:
        no_labels = True
    dendrogram(linkage_matrix, no_labels = no_labels, labels=labels, orientation="right
    plt.show()
```

Leghemoglobin data set

In [8]:

```
figure(figsize=(10,6))
build_tree(leg_data)
#Very large numbers, hence they are super similar
```

```
C:\Users\truon\anaconda3\lib\site-packages\numpy\core\_asarray.py:83: VisibleDeprecation
Warning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of l
ists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```



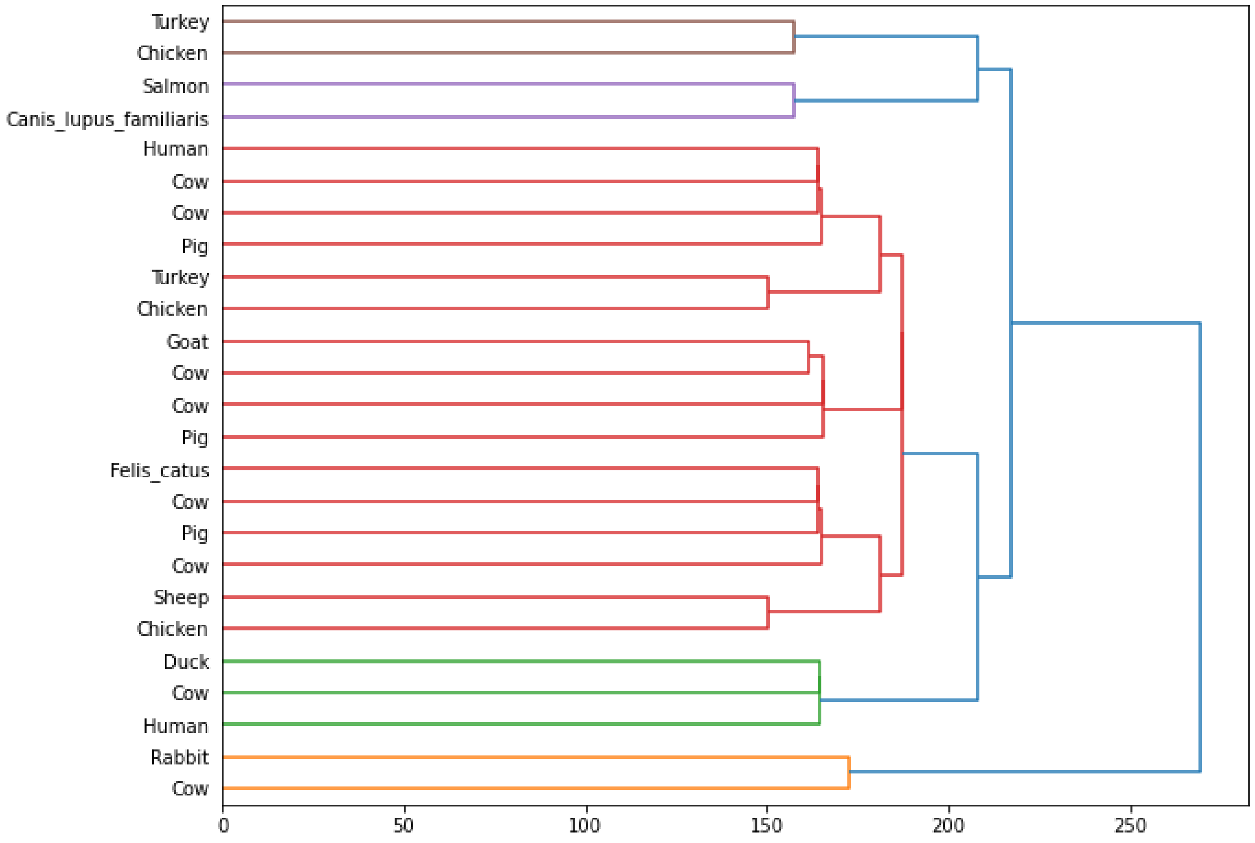
Myoglobin data set

Gather animals for figure label

```
In [9]: myo_labels = []
        for entry in myo_data:
            myo_labels.append(entry.animal)
```

Plot figure

```
In [10]: figure(figsize=(10,8))
         build_tree(myo_data, myo_labels)
         # No correlation, hence their isn't really a difference
```



Combination of both sets

Combine data

```
In [11]:  all_data = myo_data + leg_data
```

Remove outliers (Comment to view original results)

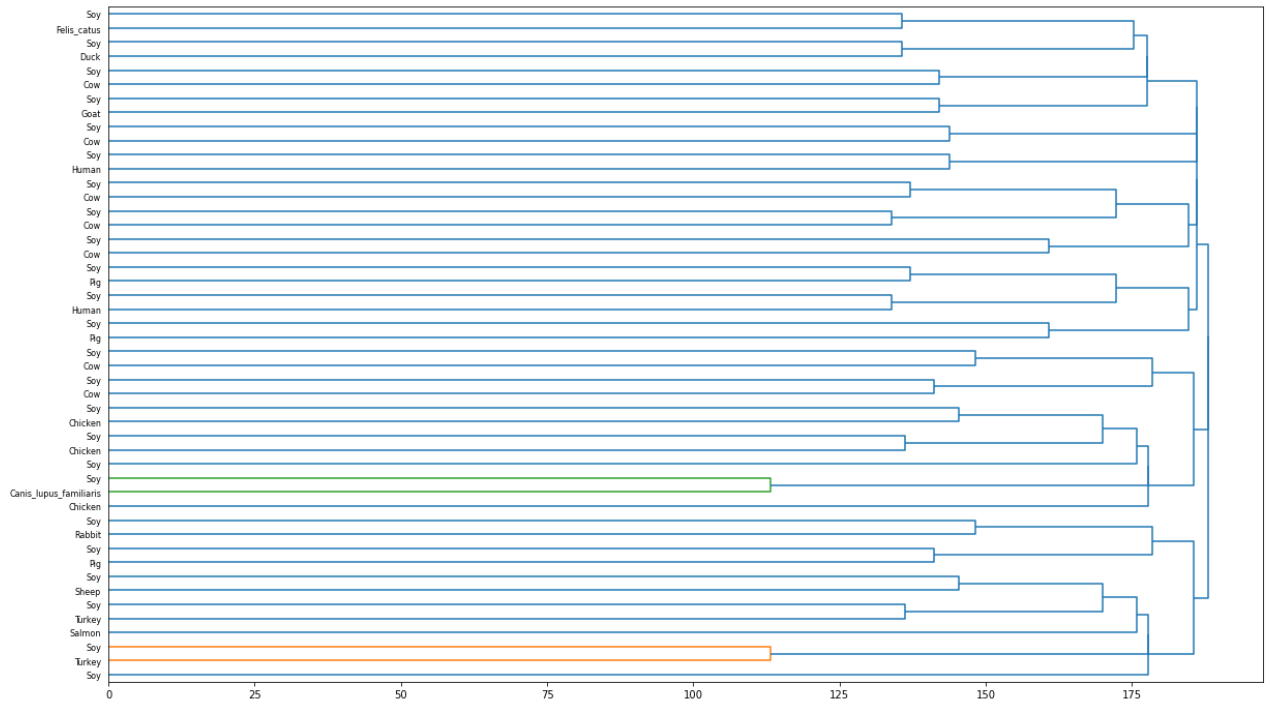
```
In [12]:  all_data = np.delete(all_data, [4,45]).tolist()
```

Gather animals for figure label

```
In [13]:  all_labels = []
          for entry in all_data:
              all_labels.append(entry.animal)
```

Plot figure

```
In [14]:  figure(figsize=(20,12))
          build_tree(all_data, all_labels)
          #Generally random/low differences; roughly every soy's closest neighbor is non-soy
```



*Align each combined file extrenally (ClustalOmega: https://www.ebi.ac.uk/Tools/msa/clustalo/)

Align X_data_unaligned.fasta and save results onto X_data_aligned.clustal

Create a combination both of files

In [15]:
```
file = open("Alignment\combined_data_unaligned.fasta", "w")
leg = open("Alignment\leg_data_unaligned.fasta", "r")
myo = open("Alignment\myo_data_unaligned.fasta", "r")

file.write(leg.read())
file.write(myo.read())

leg.close()
myo.close()
file.close()
```

Import extrenal alignment of sequences (should be included with project)

Leghemoglobin data

In [42]:
```
leg_data_aligned = AlignIO.read(open("Alignment\leg_data_aligned.clustal"), "clustal")
for entry in leg_data_aligned:
    print(entry.seq+" |(Soy)"+entry.id)
# Very clean alignment (low indels), with the exception of the first 2 insertions
```

```
MGAFTEKQEALVNSSFEAFKANLPHHSVVFFNSILEKAPAAKNMFSFLGDAVDPKNPKLAGHAEKLFGLVRDSAVQLQTKGLVVADAT
LGPIHTQKGVTDLQFAVVKEALLKTIKEAVGDKWSEELSNPWEVAYDEIAAAIKKAMAIGSLV |(Soy)V00451.1
MGAFTEKQEALVNSSFEAFKANLPHHSVVFFNSILEKAPAAKNMFSFLGDAVDPKNPKLAGHAEKLFGLVRDSAVQLQTKGLVVADAT
LGPIHTQKGVTDLQFAVVKEALLKTIKEAVGDKWSEELSNAWEVAYDEIAAAIKKAMAIGSLV |(Soy)KHN37872.1
-GAFTEKQDALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSIHAQKAVTNPEFV-VKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)pir||GPSYC2
-VAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)pdb|1FSL|A
-VAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
```

```
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)pdb|1FSL|B
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)V00453.1
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)NP_001235928.1
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)KHN37870.1
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)sp|P02238|LGBA_SOY
BN
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKASGTVVADAA
LGSVHAQKAVTDPQFVVVKEALLKTIKAAVGDKWSDELSRAWEVAYDELAAAIKKA------- |(Soy)CAA23731.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKTNGTVVADAA
LVSIHAQKAVTDPQFVVVKEALLKTIKEAVGGNWSDELSSAWEVAYDELAAAIKKA------- |(Soy)V00452.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKTNGTVVADAA
LVSIHAQKAVTDPQFVVVKEALLKTIKEAVGGNWSDELSSAWEVAYDELAAAIKKA------- |(Soy)NP_001345001.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKTNGTVVADAA
LVSIHAQKAVTDPQFVVVKEALLKTIKEAVGGNWSDELSSAWEVAYDELAAAIKKA------- |(Soy)sp|P02235|LGB1_SOY
BN
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDPTNPKLTGHAEKLFALVRDSAGQLKTNGTVVADAA
LVSIHAQKAVTDPQFVVVKEALLKTIKEAVGGNWSDELSSAWEVAYDELAAAIKKA------- |(Soy)KHN37871.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)NM_001248319.3
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)NP_001235248.2
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)KHN00941.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)AAA33980.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)sp|P02236|LGB2_SOY
BN
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)sp|P02236.2|LGB2_S
OYBN
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDPSNPKLTGHAEKLFGLVRDSAGQLKANGTVVADAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)J01301.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDPTNPKLTGHAEKLFGLVRDSAGQLKASGTVVIDAA
LGSIHGQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)NP_001235423.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDPTNPKLTGHAEKLFGLVRDSAGQLKASGTVVIDAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKA------- |(Soy)V00454.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDPTNPKLTGHAEKLFGLVRDSAGQLKASGTVVIDAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)CAA23732.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDPTNPKLTGHAEKLFGLVRDSAGQLKASGTVVIDAA
LGSIHAQKAITDPQFVVVKEALLKTIKEAVGDKWSDELSSAWEVAYDELAAAIKKAF------ |(Soy)sp|P02237.2|LGB3_S
OYBN
```

Myoglobin data

In [17]:

```python
myo_data_aligned = AlignIO.read(open("Alignment\myo_data_aligned.clustal"), "clustal")
for entry in myo_data_aligned:
    for fasta in myo_data:
        if fasta.name.split(" ")[0] == entry.id:
            print(entry.seq+" |"+fasta.animal)
            break
# Very clean alignment (mostly mismatches, less indels)
```

```
MVLSAADKGNVKAAWGKVGGHAAEYGAEALERMFLSFPTTKTYFPHFDL------SHGSAQVKGHGAKVAAALTKAVEHLDDLPGALS
ELSDLHAHKLRVDPVNFKLLSHSLLVTLASHLPSDFTPAVHASLDKFLANVSTVLTSKYR------ |Cow
----MANYDMVLQCWEPVEADYNNHGGLVLSRLFAEHPETLTLFPKFAGIAAG-DLSGNAAVAAHGATVLRKLGELLNARGDHAATLK
SLATTHANKHKIPLKNFTLITNIICKVMGEKAGL--DEAGQEALRQVMGVIIADINVTYMELGFAG |Salmon
MGLSDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQLGKILKQKGNHESELK
PLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG |Chicken
MGLSDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQLGKILKQKGNHESELK
```

```
PLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG |Chicken
MGLSDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQLGKILKQKGNHESELK
PLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG |Chicken
MGLSDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQLGKILKQKGNHESELK
PLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG |Turkey
MGLSDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQLGKILKQKGNHESELK
PLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG |Turkey
MGLSDQEWQQVLTIWGKVEADLAGHGHAVLMRLFQDHPETLDRFEKFKGLKTPDQMKGSEDLKKHGVTVLTQLGKILKQKGNHEAELK
PLAQTHATKHKIPVKYLEFISEVIIKVIAEKHSADFGADSQAAMKKALELFRNDMASKYKEFGFQG |Duck
MGLSDGEWQLVLNAWGKVEAGVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILEKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQGAMSKALELFRNDMAAQYKVLGFQG |Sheep
MGLSDGEWTLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTGAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQGAMSKALELFRNDMAAQYKVLGFQG |Goat
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDWEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDWEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTALGGILKKKGHHEAEVK
HLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG |Cow
MGLSDGEWQLVLNIWGKVETDLAGHGQEVLIRLFKNHPETLDKFDKFKHLKTEDEMKGSEDLKKHGNTVLTALGGILKKKGHHEAELK
PLAQSHATKHKIPVKYLEFISDAIIQVLQSKHSGDFHADTEAAMKKALELFRNDIAAKYKELGFQG |Canis_lupus_familiar
is
MGLSDGEWQLVLNVWGKVETDLAGHGQEVLISLFKGHPETLEKFEKFKHLKTEDEMKGSEDLKKHGSTVLTALGGILKKKGQHEAELK
PLAQSHATKHKIPVKYLEFISEAIIHVLQSKHPHDFGTDAQAAMRKALELFRNDIAAKYKELGFQG |Felis_catus
MGLSDAEWQLVLNVWGKVEADLAGHGQEVLIRLFHTHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTALGAILKKKGHHEAEIK
PLAQSHATKHKIPVKYLEFISEAIIHVLHSKHPGDFGADAQAAMSKALELFRNDIAAQYKELGFQG |Rabbit
MGLSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGATVLTALGGILKKKGHHEAEIK
PLAQSHATKHKIPVKYLEFISECIIQVLQSKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG |Human
MGLSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGATVLTALGGILKKKGHHEAEIK
PLAQSHATKHKIPVKYLEFISECIIQVLQSKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG |Human
MGLSDGEWQLVLNVWGKVEADVAGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTALGGILKKKGHHEAELT
PLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFGADAQGAMSKALELFRNDMAAKYKELGFQG |Pig
MGLSDGEWQLVLNVWGKVEADVAGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTALGGILKKKGHHEAELT
PLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFGADAQGAMSKALELFRNDMAAKYKELGFQG |Pig
MGLSDGEWQLVLNVWGKVEADVAGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTALGGILKKKGHHEAELT
PLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFGADAQGAMSKALELFRNDMAAKYKELGFQG |Pig
```

All data

In [18]:

```python
combined_data_aligned = AlignIO.read(open("Alignment\combined_data_aligned.clustal"), "
i=0
for entry in combined_data_aligned:
    is_leg = True
    for fasta in myo_data:
        if fasta.name.split(" ")[0] == entry.id:
            print(entry.seq+" |"+fasta.animal)
            is_leg = False
            i+=1
            break
    if is_leg:
        print(entry.seq+" |(Soy)"+entry.id)
# There is a very noticable segregation between leghemoglobin and hemoglobin
```

```
MGAFTEKQEALVNSSFEAFKANLPHHSVVFFNSILEKAPAAKNMFSFLGDAVDP----KNPKLAGHAEKLFGLVRDSAVQLQTKGLVV
A-DATLGPIHTQKGV-TDLQFAVVKEALLKTIKEAVGDKWSEELSNPWE----VAYDEIAAAIKKAMAIGSLV |(Soy)V00451.1
MGAFTEKQEALVNSSFEAFKANLPHHSVVFFNSILEKAPAAKNMFSFLGDAVDP----KNPKLAGHAEKLFGLVRDSAVQLQTKGLVV
```

```
A-DATLGPIHTQKGV-TDLQFAVVKEALLKTIKEAVGDKWSEELSNAWE----VAYDEIAAAIKKAMAIGSLV |(Soy)KHN3787
2.1
-GAFTEKQDALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSIHAQKAV-TNPEFV-VKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)pir||GPS
YC2
-VAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)pdb|1FSL
|A
-VAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)pdb|1FSL
|B
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)V00453.1
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)NP_00123
5928.1
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)KHN3787
0.1
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)sp|P0223
8|LGBA_SOYBN
MVAFTEKQDALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKASGTVV
A-DAALGSVHAQKAV-TDPQFVVVKEALLKTIKAAVGDKWSDELSRAWE----VAYDELAAAIKKA------- |(Soy)CAA2373
1.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKTNGTVV
A-DAALVSIHAQKAV-TDPQFVVVKEALLKTIKEAVGGNWSDELSSAWE----VAYDELAAAIKKA------- |(Soy)V00452.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKTNGTVV
A-DAALVSIHAQKAV-TDPQFVVVKEALLKTIKEAVGGNWSDELSSAWE----VAYDELAAAIKKA------- |(Soy)NP_00134
5001.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYNSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKTNGTVV
A-DAALVSIHAQKAV-TDPQFVVVKEALLKTIKEAVGGNWSDELSSAWE----VAYDELAAAIKKA------- |(Soy)sp|P0223
5|LGB1_SOYBN
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLANGVDP----TNPKLTGHAEKLFALVRDSAGQLKTNGTVV
A-DAALVSIHAQKAV-TDPQFVVVKEALLKTIKEAVGGNWSDELSSAWE----VAYDELAAAIKKA------- |(Soy)KHN3787
1.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)NM_00124
8319.3
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)NP_00123
5248.2
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)KHN0094
1.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)AAA3398
0.1
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)sp|P0223
6|LGB2_SOYBN
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)sp|P0223
6.2|LGB2_SOYBN
MGAFTEKQEALVSSSFEAFKANIPQYSVVFYTSILEKAPAAKDLFSFLSNGVDP----SNPKLTGHAEKLFGLVRDSAGQLKANGTVV
A-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)J01301.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDP----TNPKLTGHAEKLFGLVRDSAGQLKASGTVV
I-DAALGSIHGQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)NP_00123
5423.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDP----TNPKLTGHAEKLFGLVRDSAGQLKASGTVV
I-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKA------- |(Soy)V00454.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDP----TNPKLTGHAEKLFGLVRDSAGQLKASGTVV
I-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)CAA2373
2.1
MGAFTDKQEALVSSSFEAFKTNIPQYSVVFYTSILEKAPVAKDLFSFLANGVDP----TNPKLTGHAEKLFGLVRDSAGQLKASGTVV
```

```
I-DAALGSIHAQKAI-TDPQFVVVKEALLKTIKEAVGDKWSDELSSAWE----VAYDELAAAIKKAF------ |(Soy)sp|P0223
7.2|LGB3_SOYBN
MVL-SAADKGNVKAAWGKVGGHAAEYGAEALERMFLSFPTTKTYFPHFDL------SHGSAQVKGHGAKVAAAL---TKAVEHLDDLP
GALSELSDLHAHKLRVDPVNFKLLSHSLLVTLASHLPSDFTPAVHASLDKFLANVSTVLTSKYR--------- |Cow
-----MANYDMVLQCWEPVEADYNNHGGLVLSRLFAEHPETLTLFPKFAGIAAG-DLSGNAAVAAHGATVLRKL---GELLNARGDHA
ATLKSLATTHANKHKIPLKNFTLITNIICKVMGEKAGL--DEAGQEALRQVMGVIIADINVTYMELGFAG--- |Salmon
MGL-SDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQL---GKILKQKGNHE
SELKPLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG--- |Chicken
MGL-SDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQL---GKILKQKGNHE
SELKPLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG--- |Chicken
MGL-SDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQL---GKILKQKGNHE
SELKPLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG--- |Chicken
MGL-SDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQL---GKILKQKGNHE
SELKPLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG--- |Turkey
MGL-SDQEWQQVLTIWGKVEADIAGHGHEVLMRLFHDHPETLDRFDKFKGLKTPDQMKGSEDLKKHGATVLTQL---GKILKQKGNHE
SELKPLAQTHATKHKIPVKYLEFISEVIIKVIAEKHAADFGADSQAAMKKALELFRNDMASKYKEFGFQG--- |Turkey
MGL-SDQEWQQVLTIWGKVEADLAGHGHAVLMRLFQDHPETLDRFEKFKGLKTPDQMKGSEDLKKHGVTVLTQL---GKILKQKGNHE
AELKPLAQTHATKHKIPVKYLEFISEVIIKVIAEKHSADFGADSQAAMKKALELFRNDMASKYKEFGFQG--- |Duck
MGL-SDGEWQLVLNAWGKVEAGVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILEKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQGAMSKALELFRNDMAAQYKVLGFQG--- |Sheep
MGL-SDGEWTLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTGAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQGAMSKALELFRNDMAAQYKVLGFQG--- |Goat
MGL-SDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDGEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDWEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDWEWQLVLNAWGKVEADVAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AEVKHLAESHANKHKIPVKYLEFISDAIIHVLHAKHPSDFGADAQAAMSKALELFRNDMAAQYKVLGFHG--- |Cow
MGL-SDGEWQLVLNIWGKVETDLAGHGQEVLIRLFKNHPETLDKFDKFKHLKTEDEMKGSEDLKKHGNTVLTAL---GGILKKKGHHE
AELKPLAQSHATKHKIPVKYLEFISDAIIQVLQSKHSGDFHADTEAAMKKALELFRNDIAAKYKELGFQG--- |Canis_lupus_f
amiliaris
MGL-SDGEWQLVLNVWGKVETDLAGHGQEVLISLFKGHPETLEKFEKFKHLKTEDEMKGSEDLKKHGSTVLTAL---GGILKKKGQHE
AELKPLAQSHATKHKIPVKYLEFISEAIIHVLQSKHPHDFGTDAQAAMRKALELFRNDIAAKYKELGFQG--- |Felis_catus
MGL-SDAEWQLVLNVWGKVEADLAGHGQEVLIRLFHTHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTAL---GAILKKKGHHE
AEIKPLAQSHATKHKIPVKYLEFISEAIIHVLHSKHPGDFGADAQAAMSKALELFRNDIAAQYKELGFQG--- |Rabbit
MGL-SDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGATVLTAL---GGILKKKGHHE
AEIKPLAQSHATKHKIPVKYLEFISECIIQVLQSKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG--- |Human
MGL-SDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGATVLTAL---GGILKKKGHHE
AEIKPLAQSHATKHKIPVKYLEFISECIIQVLQSKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG--- |Human
MGL-SDGEWQLVLNVWGKVEADVAGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AELTPLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFGADAQGAMSKALELFRNDMAAKYKELGFQG--- |Pig
MGL-SDGEWQLVLNVWGKVEADVAGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AELTPLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFGADAQGAMSKALELFRNDMAAKYKELGFQG--- |Pig
MGL-SDGEWQLVLNVWGKVEADVAGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGNTVLTAL---GGILKKKGHHE
AELTPLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFGADAQGAMSKALELFRNDMAAKYKELGFQG--- |Pig
```

Split data 80/20

Leghemoglobin data

In [19]:
```python
leg_seqs = []
for entry in leg_data_aligned:
    leg_seqs.append(str(entry.seq))

leg_train = leg_seqs[:int(0.8*len(leg_seqs))]
```

```
leg_test = leg_seqs[int(0.8*len(leg_seqs)):]

#Unalign test set (luckily all the deletes are at the end)
for i, test in enumerate(leg_test):
    leg_test[i] = test[:test.index('-')]
```

Myoglobin data (Hand pick test values)

In [20]:
```
myo_seqs = []
for entry in myo_data_aligned:
    myo_seqs.append(str(entry.seq))

selection = [5, 15, 2, 18, 24] # Cow, human, chicken, pig, turkey
myo_train = np.delete(myo_seqs, selection).tolist()
myo_test = np.array(myo_data)[selection]
myo_test_labels = ["Cow", "Human", "Chicken", "Pig", "Turkey"]

#Extract sequences out of FASTA node
for i, test in enumerate(myo_test):
    myo_test[i] = test.sequence
```

Myoglobin data (Hand pick only cow test sequences)

In [21]:
```
cow_selection = range(4,12)
cow_test = []
for i in cow_selection:
    cow_test.append(myo_data[i].sequence)

cow_selection = [0, 10, 11, 12, 13, 14, 15, 16]
cow_aligned = np.array(myo_seqs)[cow_selection]
```

HMM from scratch

Initialize final variables

In [22]:
```
E = 0.01
ins_E = 0.01
num_proteins = 20
```

Base State

In [23]:
```
class Base_State(object):
    def __init__(self):
        self.next = {} #{"name":(probability, node_object)}
```

Normal State

In [24]:
```
class State(Base_State):
    def __init__(self):
        super().__init__()
        self.attributes = {
            "F": E, "L": E, "I": E, "M": E,
            "V": E, "A": E, "T": E, "P": E,
            "S": E, "D": E, "E": E, "N": E,
            "K": E, "Q": E, "H": E, "Y": E,
            "C": E, "W": E, "R": E, "G": E
        }
```

Insert State

In [25]:
```
class InsertState(Base_State):
    def __init__(self):
        super().__init__()
        self.next["insert"] = (ins_E, self)
```

Delete State

In [26]:
```
class DeleteState(Base_State):
    def __init__(self):
        super().__init__()
```

HMM Class (Main)

In [27]:
```
class HMM:
    def __init__(self):
        self.root = State()
        self.stop = State()

# Train the model
    def fit(self, data):
        curr_state = self.root
        curr_del_state = None
        for i in range(len(data[0])):
            # Get all the chars in the column
            aas = {}
            for seq in data:
                if seq[i] in aas.keys():
                    aas[seq[i]] += 1
                else:
                    aas[seq[i]] = 1

            # Calculate the remaining probability to distribute among Normal State's At
            att_prob = 1
            not_included = num_proteins - len(aas.keys())    #Count # of aa not include
            has_gap = 0
            if '-' in aas.keys():                            #Gap is NOT an aa
                not_included += 1
                has_gap = aas['-']
            att_prob -= not_included*E                       #aa not included error tak
            att_prob /= (len(data)-has_gap)                  #Divide among each # of se

            # Create states
            next_state = State()
            next_state_prob = 1 - ins_E      #Probability of reaching the normal state
            next_delete = None
            has_delete = False
            for aa in aas:
                if not aa == '-':
                    next_state.attributes[aa] = att_prob*aas[aa]    #Db: calculated cor
                else:
                    next_delete = DeleteState()
                    curr_state.next["delete"] = (1/len(data)*aas[aa], next_delete)    #
                    next_state_prob -= 1/len(data)*aas[aa]                           # Upd
                    if curr_del_state is not None:                                   # Lin
                        curr_del_state.next["delete"] = (1/len(data)*aas[aa], next_dele
                        curr_del_state.next["normal"] = (next_state_prob+ins_E, next_st
                    curr_del_state = next_delete
                    has_delete = True
            curr_state.next["normal"] = (next_state_prob, next_state)

            # If state no delete state was created with an active delete state,
            # update curr_del_state to None and link to new state.
            if not has_delete and curr_del_state is not None:
                curr_del_state.next["normal"] = (next_state_prob+ins_E, next_state)
                curr_del_state = None

            #Add insert state
            ins_state = InsertState()
            ins_state.next["normal"] = (1-ins_E, next_state)
            curr_state.next["insert"] = (ins_E, ins_state)

            #Iterate to next state
            curr_state = next_state

        # Link to stop node (no need to range(-1) in loop)
        ins_state = InsertState()
        ins_state.next["normal"] = (1-ins_E, self.stop)

        curr_state.next["insert"] = (ins_E, ins_state)
        curr_state.next["normal"] = (1-ins_E, self.stop)

        if curr_del_state is not None:
            curr_del_state.next["normal"] = (1, self.stop)

# Get the Viterbi Score
    def predict(self, sequence):
        # Initialize the table with initial probability (Rows: normal = 0, insert = 1,
        table = []
        for i in range(3):
            table.append([1/3])

        delete = False
        # Initialize states
        states = [None,None,None]     #Again: normal = 0, insert = 1, delete = 2)
        states[0] = self.root  #Normal
        states[1] = states[0].next["insert"][1]
        states[2] = None
        if "delete" in states[0].next.keys():
            states[2] = states[0].next["delete"][1]
            delete = True

        # Fill table
        reached_end = False
        for x in range(len(sequence)):
            next_state = None

            # If reached end of linked list and insertion is needed
            if reached_end:
                table[0].append(table[0][x] * ins_E)
                table[1].append(table[0][x] * ins_E)
                table[2].append(table[0][x] * ins_E)   # Not possible, but we need some
            else:
                next_state = states[0].next["normal"][1]

                # Calculate value for the normal state row
                temp = []
                for i in range(3):
                    if i == 2 and (delete or states[i] == None):   # In case delete sta
                        break
                    temp.append(table[i][x] * states[i].next["normal"][0] * next_state.
                normal_value = max(temp)
                table[0].append(normal_value)

                # Calculate value for the insert state row
                temp = []
                for i in range(2):   # An insert in a delete (delete->insert) is not po
                    temp.append(table[i][x] * states[i].next["insert"][0])
                insert_value = max(temp)
                table[1].append(insert_value)

                # Calculate value for the delete state row
                # Also: A delete in an insert (insert->delete) is not possible: would j
                temp = []
                delete_value = table[0][x] * ins_E     # Psuedo-delete state if delete
                if "delete" in states[0].next:    # If delete state exists
                    temp.append(table[0][x] * states[0].next["delete"][0])

                    if not states[2] == None and "delete" in states[2].next.keys() and
                        temp.append(table[2][x] * states[2].next["delete"][0])
                    else:
                        temp.append(table[2][x] * ins_E) # Psuedo-delete state
                    delete_value = max(temp)
                table[2].append(delete_value)

            #Iterate nodes
            if reached_end or len(next_state.next) == 0:    # Sequence is longer than t
                reached_end = True
            else:
                # Handle delete state. If there is a delete, has to skip an iteration b
                if "delete" in next_state.next.keys():
                    states[2] = next_state.next["delete"][1]
                    delete = True
                else:
                    if delete:
                        delete = False
                    else:
                        states[2] = None
                #Iterate nornal state
                if isinstance(next_state, DeleteState):
                    states[1] = None
                else:
                    states[1] = next_state.next["insert"][1]
                states[0] = next_state

        # Get Viterbi score
        score = max(
            table[0][-1],
            table[1][-1],
            table[2][-1]
        )

        return score
```

Train Model and Test

In [28]:
```
leg_hmm = HMM()
leg_hmm.fit(leg_train)
```

```
myo_hmm = HMM()
myo_hmm.fit(myo_train)
```

Test Leghemoglobin HMM

In [29]:
```
for test in leg_test:
    score = leg_hmm.predict(test)
    print("Soy\t"+str(score))
# Base scores to sompare with below
```

```
Soy     5.30033873848049e-20
Soy     5.614113567991492e-29
Soy     7.457788463719895e-27
Soy     4.101783655045943e-27
Soy     4.101783655045943e-27
```

Test Myohemoglobin HMM

In [30]:
```
for test, label in zip(myo_test, myo_test_labels):
    score = myo_hmm.predict(test)
    print(label+"\t"+str(score))
# Base scores to sompare with below
```

```
Cow     5.746529244269801e-29
Human   2.404579744105982e-35
Chicken 7.613327645334508e-43
Pig     4.477347112747736e-31
Turkey  7.613327645334508e-43
```

Use Leghemoglobin test set on Myohemoglobin HMM

In [31]:
```
for test in leg_test:
    score = myo_hmm.predict(test)
    print("Soy\t"+str(score))
# Bad scores
```

```
Soy     4.4437712440684957e-262
Soy     6.7600268408003205e-264
Soy     6.76002684080032e-262
Soy     6.7600268408003205e-264
Soy     6.7600268408003205e-264
```

Use Myohemoglobin test set on Leghemoglobin HMM

In [32]:
```
for test, label in zip(myo_test, myo_test_labels):
    score = leg_hmm.predict(test)
    print(label+"\t"+str(score))
# Bad scores
```

```
Cow     2.829675374206272e-281
Human   2.829675374206272e-281
Chicken 2.187259222762203e-283
Pig     2.2971304687806525e-280
Turkey  2.187259222762203e-283
```

Extra analysis: Cow Myohemoglobin test (359 lines of code up to here)

Use Cow only Myohemoglobin test set on Leghemoglobin HMM

In [33]:
```
for test in cow_test:
    score = leg_hmm.predict(test)
    print("Cow\t"+str(score))
# Bad scores
```

```
Cow     5.929022975528739e-267
Cow     2.829675374206272e-281
Cow     2.829675374206272e-281
Cow     2.829675374206272e-281
Cow     2.829675374206272e-281
Cow     2.829675374206272e-281
Cow     2.829675374206272e-281
Cow     2.829675374206272e-281
```

Train model with Cow sequences only and test with *all* Leghemoglobin sequences

In [34]:

```
cow_hmm = HMM()
cow_hmm.fit(cow_aligned)
for entry in leg_data:
    score = cow_hmm.predict(entry.sequence)
    print("Soy\t"+str(score))
# Bad scores
```

```
Soy     8.891572360514127e-253
Soy     8.891572360514127e-253
Soy     1.2760140602112642e-266
Soy     1.7963806148048626e-266
Soy     1.8228772288732357e-265
Soy     1.276014060211264e-264
Soy     1.2790445936042654e-275
Soy     1.8228772288732357e-265
Soy     1.8228772288732357e-265
Soy     1.8228772288732357e-265
Soy     1.276014060211264e-264
Soy     1.276014060211264e-264
Soy     8.958808362748201e-253
Soy     1.2760140602112642e-266
Soy     1.2760140602112642e-266
Soy     1.2790445936042654e-275
Soy     1.7963806148048626e-268
Soy     1.7963806148048626e-268
Soy     1.8228772288732357e-265
Soy     1.2760140602112642e-266
Soy     1.276014060211264e-264
Soy     1.2760140602112642e-266
Soy     1.2760140602112642e-266
Soy     1.7963806148048626e-268
Soy     1.2760140602112642e-266
```

Test *all* myhemoglobin sequences on Leghemoglobin model

In [35]:

```
for entry in myo_data:
    #Formating
    t = ""
    if len(entry.animal) < 12:
        t = "\t"
        if len(entry.animal) < 8:
            t += "\t"

    #Calucation here
    score = leg_hmm.predict(entry.sequence)
    print(entry.animal+"\t"+t+str(score))
# Soy is most similar to Salmon and flawed Cow
```

```
Canis_lupus_familiaris  2.829675374206272e-281
```

```
Chicken                 2.187259222762203e-283
Chicken                 2.187259222762203e-283
Chicken                 2.187259222762203e-283
Cow                     5.929022975528739e-267
Cow                     2.829675374206272e-281
Cow                     2.829675374206272e-281
Cow                     2.829675374206272e-281
Cow                     2.829675374206272e-281
Cow                     2.829675374206272e-281
Cow                     2.829675374206272e-281
Cow                     2.829675374206272e-281
Duck                    1.753963170733009e-281
Felis_catus             2.829675374206272e-281
Goat                    2.829675374206272e-281
Human                   2.829675374206272e-281
Human                   2.829675374206272e-281
Pig                     2.2971304687806525e-280
Pig                     2.2971304687806525e-280
Pig                     2.2971304687806525e-280
Rabbit                  3.8729782571052977e-283
Salmon                  2.919629763048611e-269
Sheep                   3.8729782571052977e-283
Turkey                  2.187259222762203e-283
Turkey                  2.187259222762203e-283
```

Test *all* myhemoglobin sequences on Cow model

In [36]:
```python
for entry in myo_data:
    #Formating
    t = ""
    if len(entry.animal) < 12:
        t = "\t"
        if len(entry.animal) < 8:
            t += "\t"

    #Calucation here
    score = cow_hmm.predict(entry.sequence)
    print(entry.animal+"\t"+t+str(score))
# Notice: one flawed Cow and Salmon
```

```
Canis_lupus_familiaris  2.0711225011059564e-64
Chicken                 1.0356012715263339e-91
Chicken                 1.0356012715263339e-91
Chicken                 1.0356012715263339e-91
Cow                     1.0774767495508747e-200
Cow                     3.4790542839395505e-22
Cow                     1.3916217135758214e-22
Cow                     3.4790542839395505e-22
Cow                     1.3916217135758214e-22
Cow                     3.4790542839395505e-22
Cow                     3.4790542839395505e-22
Cow                     3.4790542839395505e-22
Duck                    1.2726626584187439e-94
Felis_catus             1.4711700905980877e-62
Goat                    1.6297455850862154e-29
Human                   1.4901012337395633e-63
Human                   1.4901012337395633e-63
Pig                     2.1608344133386394e-52
Pig                     2.1608344133386394e-52
Pig                     2.1608344133386394e-52
Rabbit                  3.365530858479962e-53
Salmon                  8.929123017613468e-281
Sheep                   1.425291884884441e-28
```

```
Turkey                  1.0356012715263339e-91
Turkey                  1.0356012715263339e-91
```

Export Train/Test set to compare with HMM given from Lab (393 lines of code up to here)

Leghemoglobin Training Set

In [37]:
```
file = open("Export\export_leg_train.fa","w")
for i, train in enumerate(leg_train):
    file.write(">Sequence (Soy) "+str(i+1)+"\n")
    file.write(train+"\n")
file.close()
```

Leghemoglobin Test Set

In [38]:
```
file = open("Export\export_leg_test.fa","w")
for i, train in enumerate(leg_test):
    file.write(">Sequence (Soy) "+str(i+1)+"\n")
    file.write(train+"\n")
file.close()
```

Myoglobin Test Set

In [39]:
```
file = open("Export\export_myo_train.fa","w")
for i, train in enumerate(myo_train):
    file.write(">Sequence "+str(i+1)+"\n")
    file.write(train+"\n")
file.close()
```

Myoglobin Test Set

In [40]:
```
file = open("Export\export_myo_test.fa","w")
for i, (train, name) in enumerate(zip(myo_test, myo_test_labels)):
    file.write(">"+name+"\n")
    file.write(train+"\n")
file.close()
```

Cow only Dataset

In [41]:
```
file = open("Export\export_cow_test.fa","w")
for i, train in enumerate(cow_test):
    file.write(">"+str(i)+"\n")
    file.write(train+"\n")
file.close()
```

In [ ]: