BBC - Laboratoire 2

Vincent Guidoux

Alignement de séquences et arbres phylogénétiques

- Professeur: Carlos Peña (carlos.pena@heig-vd.ch (mailto:carlos.pena@heig-vd.ch))
- Assistant: Xavier Brochet (xavier.brochet@heig-vd.ch (mailto:xavier.brochet@heig-vd.ch))

Date: Printemps 2019

Objectifs pédagogiques

Pour réaliser ce laboratoire, du vocabulaire et des notions de base en biologie seront introduits ou perfectionnés (revoir Labo-1 si nécessaire). A la fin du laboratoire, l'étudiant devrait:

- Etre à l'aise avec l'utilisation d'Entrez Direct pour accéder à Genbank.
- Savoir utiliser des outils d'alignement de séquence, à l'aide du logiciel Aliview et avec la librairie biopython.
- Savoir construire un arbre phylogénétique à partir d'un alignement et l'interpréter.

But du labo

La septicémie hémorragique virale est une maladie très contagieuse et mortelle qui touche les poissons de type salmonidés (les humains ne sont pas touchés). Elle est causée par un virus qui engendre chaque année de grosses pertes économiques pour les pisciculteurs. Votre but est de visualiser comment le virus se propage dans le monde, en comparant les séquences de la protéine G (glycoprotéine se trouvant à la surface du virus) de plusieurs isolats.

Démarche

1. Visualisation de la présence du virus dans le monde (Genbank, Entrez Direct et biopython).
2. Alignement des séquences protéiques pour pouvoir les comparer (MUSCLE, (ClustalW), Clustal Omega).
3. Construction de l'arbre phylogénétique et clustering hiérarchique pour identifier des groupes d'isolats similaires.

Rapport de labo

Merci de répondre aux questions posées sur fond cyan de façon concise dans la cellule située juste en dessous, et de nous retourner ce notebook.

0. Ceci est une question.

1. Le virus dans le monde - visualisation d'isolats séquencés

Comme lors du premier labo, vous allez apprendre à utiliser certains outils à la fois en mode graphique et en mode ligne de commande.

Votre premier objectif est de récupérer sur Genbank la localisation géographique de toutes les souches séquencées du virus (protéine G uniquement) et de les visualiser sur une carte du monde.

→ **Depuis votre navigateur, allez sur le site de [Genbank](http://www.ncbi.nlm.nih.gov/genbank/) et dans la barre de recherche, tapez**: G[gene] AND VHSV[orgn] NOT "mRNA"[title] AND "complete cds"[title]

Cette requête limite les recherches aux entrées dont le nom du gène s'intitule G, dont l'organisme est le VHSV (Viral Hemorrhagic Septicemia Virus), dont le titre ne contient pas le mot "mRNA" et dont la séquence est complète.

→ **Cliquez sur la première entrée: https://www.ncbi.nlm.nih.gov/nuccore/MH836523.1**

Dans la partie FEATURES - source, vous pouvez voir que le pays d'origine est Islande.

→ **Cliquez sur cette entrée: http://www.ncbi.nlm.nih.gov/nuccore/341904092** Vous pouvez voir qu'en plus du pays, la latitude et la longitude sont également indiqués.

1.1 Quelles sont les coordonnées (latitude, longitude) et pays d'origine de l'isolat dont l'ACCESSION=HQ112247 et gi=341904064 (attention le gi n'est officiellement plus utilisé)?

Originaire de Finlande aux latitudes et longitudes : 60.24 N 22.06 E

Votre recherche avec la requête initiale a retourné 390 entrées. Afin de récupérer les données géographiques de tous les isolats, il vous faut avoir recours à un langage de script. Comme vu au Labo-1, nous ferons appel à Entrez Direct en utilisant la librairie biopython.

→ **Importez les modules suivants**

In [1]:
```
from Bio import Entrez
Entrez.email = "vincent.guidoux@heig-vd.ch" # une adresse email valide est néces
saire
from mpl_toolkits.basemap import Basemap # pour dessiner une carte du monde
import matplotlib.pyplot as plt # pour générer des graphiques
%matplotlib inline
#problème avec le module mpl_toolkits.basemap..
#esssai pour le récupérer
#j'ai mis a jour:
#pip3.6 install --upgrade biopython --> pas fonctionné
#pip3 install --upgrade jupyter matplotlib numpy pandas scipy scikit-learn six C
ollecting jupyter
#pip3.6 install notebook --upgrade
#conda install -c anaconda biopython
```

→ **Commencez par récupérer les identifiants gi correspondant à la requête en utilisant Entrez.esearch()**

In [2]:
```
query = 'G[gene] AND VHSV[orgn] NOT "mRNA"[title] AND "complete cds"[title]'
handle = Entrez.esearch(db='nucleotide',retmax=1000,term=query)
record = Entrez.read(handle)
gi = record['IdList']
#print(gi)
print(len(gi))
```

```
390
```

→ Allez chercher les informations Genbank sur ces séquences en utilisant Entrez.efetch()

In [3]:
```
protein_info = Entrez.efetch(db="nucleotide",id=gi,rettype="gb",retmode="xml")
protein_info = Entrez.read(protein_info)
print(len(protein_info))
```

```
390
```

`protein_info` est une liste de 390 éléments. Chaque élément consiste en une structure de listes et dictionnaires contenant toute l'information de Genbank correpondant à la requête.

→ Imprimez le dixième élément de protein_info et quelques unes de ses caractéristiques.

In [4]:
```
print(protein_info[10])
print('')
print('')
print(protein_info[10].keys())
print('')
print('')
features = protein_info[10]['GBSeq_feature-table'][0]['GBFeature_quals']
# features = protein_info[60]['GBSeq_feature-table'][0]['GBFeature_quals']
print(features)
```

DictElement({'GBSeq_locus': 'KU728260', 'GBSeq_length': '1524', 'GBSeq_strande
dness': 'single', 'GBSeq_moltype': 'cRNA', 'GBSeq_topology': 'linear', 'GBSeq_
division': 'VRL', 'GBSeq_update-date': '01-MAR-2017', 'GBSeq_create-date': '01
-MAR-2017', 'GBSeq_definition': 'Viral hemorrhagic septicemia virus isolate DK
-7690 glycoprotein (G) gene, complete cds', 'GBSeq_primary-accession': 'KU7282
60', 'GBSeq_accession-version': 'KU728260.1', 'GBSeq_other-seqids': ['gb|KU728
260.1|', 'gi|1149552504'], 'GBSeq_source': 'Viral hemorrhagic septicemia virus
', 'GBSeq_organism': 'Viral hemorrhagic septicemia virus', 'GBSeq_taxonomy': '
Viruses; ssRNA viruses; ssRNA negative-strand viruses; Mononegavirales; Rhabdo
viridae; Novirhabdovirus', 'GBSeq_references': [DictElement({'GBReference_refe
rence': '1', 'GBReference_position': '1..1524', 'GBReference_authors': ['Mikke
lsen,S.S.', 'Panzarin,V.', 'Schuetze,H.', 'Skall,H.F.', 'Fusaro,A.', 'Korsholm
,H.', 'Olesen,N.-J.'], 'GBReference_title': 'Molecular tracing of viral hemorr
hagic septicemia outbreaks in Denmark', 'GBReference_journal': 'Unpublished'},
attributes={}), DictElement({'GBReference_reference': '2', 'GBReference_positi
on': '1..1524', 'GBReference_authors': ['Mikkelsen,S.S.', 'Schuetze,H.'], 'GBR
eference_title': 'Direct Submission', 'GBReference_journal': 'Submitted (18-FE
B-2016) Section for Diagnostics and Scientific Advice, DTU Veterinary Institut
e, Bulowsvej 27, Frederiksberg 1870, Denmark'}, attributes={})], 'GBSeq_commen
t': '##Assembly-Data-START## ; Sequencing Technology :: Sanger dideoxy sequenc
ing ; ##Assembly-Data-END##', 'GBSeq_feature-table': [DictElement({'GBFeature_
key': 'source', 'GBFeature_location': '1..1524', 'GBFeature_intervals': [DictE
lement({'GBInterval_from': '1', 'GBInterval_to': '1524', 'GBInterval_accession
': 'KU728260.1'}, attributes={})], 'GBFeature_quals': [DictElement({'GBQualifi
er_name': 'organism', 'GBQualifier_value': 'Viral hemorrhagic septicemia virus
'}, attributes={}), DictElement({'GBQualifier_name': 'mol_type', 'GBQualifier_
value': 'viral cRNA'}, attributes={}), DictElement({'GBQualifier_name': 'isola
te', 'GBQualifier_value': 'DK-7690'}, attributes={}), DictElement({'GBQualifie
r_name': 'host', 'GBQualifier_value': 'Oncorhynchus mykiss'}, attributes={}),
DictElement({'GBQualifier_name': 'db_xref', 'GBQualifier_value': 'taxon:11287'
}, attributes={}), DictElement({'GBQualifier_name': 'country', 'GBQualifier_va
lue': 'Denmark'}, attributes={}), DictElement({'GBQualifier_name': 'collection
_date', 'GBQualifier_value': '07-Dec-1994'}, attributes={}), DictElement({'GBQ
ualifier_name': 'note', 'GBQualifier_value': 'genotype: 1a'}, attributes={})]}
, attributes={}), DictElement({'GBFeature_key': 'gene', 'GBFeature_location':
'1..1524', 'GBFeature_intervals': [DictElement({'GBInterval_from': '1', 'GBInt
erval_to': '1524', 'GBInterval_accession': 'KU728260.1'}, attributes={})], 'GB
Feature_quals': [DictElement({'GBQualifier_name': 'gene', 'GBQualifier_value':
'G'}, attributes={})]}, attributes={}), DictElement({'GBFeature_key': 'CDS', '
GBFeature_location': '1..1524', 'GBFeature_intervals': [DictElement({'GBInterv
al_from': '1', 'GBInterval_to': '1524', 'GBInterval_accession': 'KU728260.1'},
attributes={})], 'GBFeature_quals': [DictElement({'GBQualifier_name': 'gene',
'GBQualifier_value': 'G'}, attributes={}), DictElement({'GBQualifier_name': 'c
odon_start', 'GBQualifier_value': '1'}, attributes={}), DictElement({'GBQualif
ier_name': 'transl_table', 'GBQualifier_value': '1'}, attributes={}), DictElem
ent({'GBQualifier_name': 'product', 'GBQualifier_value': 'glycoprotein'}, attr
ibutes={}), DictElement({'GBQualifier_name': 'protein_id', 'GBQualifier_value'
: 'AQT19222.1'}, attributes={}), DictElement({'GBQualifier_name': 'translation
', 'GBQualifier_value': 'MEWNTFFLVILIIIIKSTTPQITQRPPVENISTYHADWDTPLYTHPSNCRDDS
FVPIRPAQLRCPHEFEDINRGLVSVPTKIIHLPLSVTSVSAVASGHYLHRVTYRVTCSTSFFGGQTIEKTILEAKLSR
QEATDEASKDHEYPFFPEPSCIWMKNNVHKDITHYYKTPKTVSVDLYSRKFLNPDFIEGVCTTSPCQTHWQGVYWVGA
TPNAHCPTSETLEGHLFTRTHDHRVVKAIVAGHHPWGLTMACTVKFCGEDWIKTDLGDLIQVTGPGGTGKLTPNKCVN
TDVQMRGATDDFSYLNHLITNMAQRTECLDAHSDITASGKVSSFLLSKFRPSHPGPGKAHYLLDGQIMRGDCDYEAVV
SINYNSAQYKTVNNTWKSWKRVDNNTDGYDGMIFGDKLIIPDIEKYQSVYDSGMLVQRNLVEVPHLSIVFVSNTSDLS
TNHIHTNLIPSDWSFHWSLWPSLSGMGVVGGAFLLLVLCCCCKASPPTPNYGIPMQQFSRSQTV'}, attributes
={})]}, attributes={})], 'GBSeq_sequence': 'atggaatggaatactttcttcttggtgatcttga
tcatcatcataaagagcaccacaccacagatcactcaacgacctccggttgaaaacatctcgacgtaccatgcagatt
gggacactccgctatacactcatccctccaactgcagggacgattcctttgtcccgattcgaccagctcaactcaggt
gtcctcatgaatttgaagacataaacaggggactggtttccgtcccaaccaagatcatccatctcccgctatcagtca
ccagcgtctccgcagtagcgagcggccactacctgcacagagtgacttatcgagtcacctgttcgaccagcttctttg
gagggcaaaccattgaaaagaccatcttggaggcgaaactgtctcgtcaggaggccacagacgaggcaagcaaggacc
acgagtacccgttcttccctgaaccctcctgcatctggatgaaaaacaatgtccataaggacataactcactattaca
agaccccaaaaacagtatcggtggatctctacagcaggaaatttctcaaccctgatttcatcgaaggggtctgcacaa
cctcgccctgtcaaactcattggcagggagtctattgggtcggcgccacacccaatgcccattgccccacgtcggaaa
cactagaaggacacctgttcaccaggacccatgatcacagggtggtcaaggcaattgtggcaggccatcatccctggg
gactcacaatggcatgcacagtgaaattctgcggggaagattggatcaagactgacctgggagacctgatccaggtga
caggaccgggggcacggggaaactgactccaaataagtgtgtcaatactgatgtccagatgaggggggcaacagacg
actttttcttatctcaaccatctcatcaccaacatggctcaaagaaccgagtgcctagatgcccatagtgatatcaccg
cttctgggaaagtgtcctcatttctcctctcaaagtttcgtcccagccaccctggacccggcaaggcacactatcttc

1.2 Quel est le pays d'origine du 60ème isolat dans notre liste?

Danemark, pour arriver à ce résultat, j'ai exécuté la ligne `features = protein_info[60]['GBSeq_feature-table'][0]['GBFeature_quals']` , et chercher l'occurence de `country`

→ Pour extraire le pays automatiquement, faites une boucle sur les features jusqu'à tomber sur "country":

In [5]:
```
for j,feat in enumerate(features):
    if feat['GBQualifier_name']=='country':
        print("Pays d'origine: "+feat['GBQualifier_value'].split(':')[0])
```

```
Pays d'origine: Denmark
```

1.3 Généralisez ce code pour qu'il imprime le pays d'origine des 390 isolats et enregistre le résultat dans une variable (liste de 390 éléments) appelée `country` (si le pays n'est pas disponible, ajoutez le texte 'NA' (not available) à la place).

In [6]:
```python
country = []

for info in protein_info:
    current_country = 'NA'
    for j,feat in enumerate(info['GBSeq_feature-table'][0]['GBFeature_quals']):

        if feat['GBQualifier_name']=='country':
            current_country = feat['GBQualifier_value'].split(':')[0]
    country.append(current_country)

print(country)
```

```
['Iceland', 'Iceland', 'Finland', 'Finland', 'Finland', 'Finland', 'Finland',
'Finland', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', '
Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'D
enmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'De
nmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Den
mark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denm
ark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denma
rk', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmar
k', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark
', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark'
, 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark',
'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Italy', 'It
aly', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy',
'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy
', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'It
aly', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy',
'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy
', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'It
aly', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy',
'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy
', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'It
aly', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy',
'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy
', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'Italy', 'It
aly', 'Italy', 'Italy', 'Norway', 'Norway', 'Norway', 'Norway', 'Iran', 'Iran'
, 'Iran', 'Iran', 'Spain', 'NA', 'Turkey', 'Turkey', 'Turkey', 'Norway', 'Norw
ay', 'Norway', 'Norway', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'South Korea', 'South Korea', 'Denmark', 'Denmark', 'Denmark', 'De
nmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Den
mark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denm
ark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denma
rk', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmar
k', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark
', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark'
, 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark',
'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Denmark', 'Finland', '
Finland', 'Finland', 'Finland', 'Finland', 'Finland', 'Finland', 'Finland', 'F
inland', 'Finland', 'Finland', 'Finland', 'Finland', 'Finland', 'Finland', 'Fi
nland', 'Finland', 'Finland', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA', 'NA',
'NA', 'NA', 'NA', 'Norway', 'Norway', 'Norway', 'Canada', 'Canada', 'USA', 'US
A', 'Japan', 'Japan', 'USA', 'Canada', 'Canada', 'Canada', 'Slovenia', 'Norway
']
```

→ **Pour certains isolats, la latitude et la longitude sont également indiqués. Par exemple, exécutez:**

In [7]:
```
print(protein_info[294]['GBSeq_feature-table'][0]['GBFeature_quals'])
```

```
[DictElement({'GBQualifier_name': 'organism', 'GBQualifier_value': 'Viral hemo
rrhagic septicemia virus'}, attributes={}), DictElement({'GBQualifier_name': '
mol_type', 'GBQualifier_value': 'viral cRNA'}, attributes={}), DictElement({'G
BQualifier_name': 'isolate', 'GBQualifier_value': 'ka664_04'}, attributes={}),
DictElement({'GBQualifier_name': 'db_xref', 'GBQualifier_value': 'taxon:11287'
}, attributes={}), DictElement({'GBQualifier_name': 'country', 'GBQualifier_va
lue': 'Finland: Archipelago Sea'}, attributes={}), DictElement({'GBQualifier_n
ame': 'lat_lon', 'GBQualifier_value': '60.29 N 21.29 E'}, attributes={}), Dict
Element({'GBQualifier_name': 'collection_date', 'GBQualifier_value': '09-Jun-2
004'}, attributes={}), DictElement({'GBQualifier_name': 'identified_by', 'GBQu
alifier_value': 'T. Gadd'}, attributes={}), DictElement({'GBQualifier_name': '
PCR_primers', 'GBQualifier_value': 'fwd_name: G1_forward, fwd_seq: cgggcaggcga
aggacta, fwd_name: G2_forward, fwd_seq: atggaatggaatacttttttc, fwd_seq: caacct
cgccctgtcaaactcat, fwd_seq: tggacccggcaaggcacact, rev_name: G1_reverse, rev_se
q: cggagacgctggtgactgata, rev_name: G2_reverse, rev_seq: tgtgatcatgggtcctggtg,
rev_name: G3_reverse, rev_seq: gtccccaaatatcatcccatcgta, rev_name: NAH_reverse
, rev_seq: ctaggagacttatcctcatgtc'}, attributes={})]
```

Dans ce cas, `lat_lon = 60.29 N 21.29 E` (latitude 60.29°, longitude 21.29°).

1.4 Généralisez votre code pour qu'il enregistre aussi la latitude et la longitude dans deux variables séparées (mettez un `float('nan')` si elles ne sont pas disponibles). Astuce: Utilisez la fonction split(' ') pour extraire les nombres qui vous intéressent de lat_lon (à convertir en float aussi).

In [8]:
```
lat_lon = [] #J'ai mal compris la donnée, j'ai fait un dict qui regroupe les lon
gitudes et les latitudes

for info in protein_info: #Nous parcourons chaque protéine
    current_lat_lon = {'lat': float('nan'), 'lon': float('nan')}
    for j,feat in enumerate(info['GBSeq_feature-table'][0]['GBFeature_quals']):#
chaqune de ses features

        if feat['GBQualifier_name']=='lat_lon': #si la feature courante est lat_
lon, on les stocke
            current = feat['GBQualifier_value'].split('N')
            lat = current[0]
            lon = current[1].split('E')[0]
            current_lat_lon['lat'] = lat
            current_lat_lon['lon'] = lon
    lat_lon.append(current_lat_lon)

print(len(lat_lon))
```

```
390
```

Comme vous le verrez, la latitude et la longitude ne sont disponibles que pour un petit nombre d'isolats, alors que le pays d'origine est connu pour la majorité d'entre eux. Afin de pouvoir représenter les isolats sur une carte du monde, nous allons utiliser une valeur moyenne de latitude et de longitude par pays pour les isolats dont on ne connaît que le pays d'origine.

→ **Utilisez la librairie pandas pour lire le fichier countries_latlon.csv.**

> En cas de soucis d'encodage, ajoutez les lignes suivantes à ~./bash_profile et re-lancez ipython notebook:
>
> export LC_ALL=en_US.UTF-8
> export LANG=en_US.UTF-8

In [9]:
```
import pandas as pd # très utile pour lire des fichiers du type csv, tsv, etc.
country_latlon = pd.read_csv('countries_latlon.csv',sep='\t',header=0)
print(country_latlon)
```

```
    iso-code   latitude   longitude                         country
0         AD  42.546245    1.601554                         Andorra
1         AE  23.424076   53.847818            United Arab Emirates
2         AF  33.939110   67.709953                     Afghanistan
3         AG  17.060816  -61.796428             Antigua and Barbuda
4         AI  18.220554  -63.068615                        Anguilla
5         AL  41.153332   20.168331                         Albania
6         AM  40.069099   45.038189                         Armenia
7         AN  12.226079  -69.060087            Netherlands Antilles
8         AO -11.202692   17.873887                          Angola
9         AQ -75.250973   -0.071389                      Antarctica
10        AR -38.416097  -63.616672                       Argentina
11        AS -14.270972 -170.132217                  American Samoa
12        AT  47.516231   14.550072                         Austria
13        AU -25.274398  133.775136                       Australia
14        AW  12.521110  -69.968338                           Aruba
15        AZ  40.143105   47.576927                      Azerbaijan
16        BA  43.915886   17.679076          Bosnia and Herzegovina
17        BB  13.193887  -59.543198                        Barbados
18        BD  23.684994   90.356331                      Bangladesh
19        BE  50.503887    4.469936                         Belgium
20        BF  12.238333   -1.561593                    Burkina Faso
21        BG  42.733883   25.485830                        Bulgaria
22        BH  25.930414   50.637772                         Bahrain
23        BI  -3.373056   29.918886                         Burundi
24        BJ   9.307690    2.315834                           Benin
25        BM  32.321384  -64.757370                         Bermuda
26        BN   4.535277  114.727669                          Brunei
27        BO -16.290154  -63.588653                         Bolivia
28        BR -14.235004  -51.925280                          Brazil
29        BS  25.034280  -77.396280                         Bahamas
..       ...        ...         ...                             ...
215       TL  -8.874217  125.727539                      Timor-Leste
216       TM  38.969719   59.556278                     Turkmenistan
217       TN  33.886917    9.537499                          Tunisia
218       TO -21.178986 -175.198242                            Tonga
219       TR  38.963745   35.243322                           Turkey
220       TT  10.691803  -61.222503              Trinidad and Tobago
221       TV  -7.109535  177.649330                           Tuvalu
222       TW  23.697810  120.960515                           Taiwan
223       TZ  -6.369028   34.888822                         Tanzania
224       UA  48.379433   31.165580                          Ukraine
225       UG   1.373333   32.290275                           Uganda
226       UM        NaN         NaN  U.S. Minor Outlying Islands
227       US  37.090240  -95.712891                              USA
228       UY -32.522779  -55.765835                          Uruguay
229       UZ  41.377491   64.585262                       Uzbekistan
230       VA  41.902916   12.453389                     Vatican City
231       VC  12.984305  -61.287228  Saint Vincent and the Grenadines
232       VE   6.423750  -66.589730                        Venezuela
233       VG  18.420695  -64.639968          British Virgin Islands
234       VI  18.335765  -64.896335             U.S. Virgin Islands
235       VN  14.058324  108.277199                         Viet Nam
236       VU -15.376706  166.959158                          Vanuatu
237       WF -13.768752 -177.156097            Wallis and Futuna
238       WS -13.759029 -172.104629                            Samoa
239       XK  42.602636   20.902977                           Kosovo
240       YE  15.552727   48.516388                            Yemen
241       YT -12.827500   45.166244                          Mayotte
242       ZA -30.559482   22.937506                     South Africa
243       ZM -13.133897   27.849332                           Zambia
244       ZW -19.015438   29.154857                         Zimbabwe

[245 rows x 4 columns]
```

→ Pour trouver la latitude et la longitude moyenne de l'Iran par exemple, exécutez le code suivant:

In [10]:
```
idx_country = list(country_latlon['country']).index('Iran')
lat = country_latlon['latitude'][idx_country]
lon = country_latlon['longitude'][idx_country]
print('Latitude: '+str(lat))
print('Longitude: '+str(lon))
```

```
Latitude: 32.427908
Longitude: 53.68804599999999
```

1.5 (a) Généralisez votre code pour qu'il enregistre la latitude et la longitude moyenne lorsque celles-ci ne sont pas disponibles mais que vous connaissez le pays d'origine.

1.5 (b) Afin de retrouver facilement les isolats par leur accession.version (en plus du gi), définissez aussi `name=[]` et rajoutez dans votre boucle `name.append(protein_info[i]['GBSeq_accession-version'])`

In [11]:
```
name = []

for i in range(len(protein_info)): # Nous parcourons toutes les protéines
    current_lat_lon = {'lat': float('nan'), 'lon': float('nan')}
    if lat_lon[i]['lat'] != None: # Si la latitude n'est pas définie
        current_country = country[i] # Nous prenons le pays correspondant
        if current_country != 'NA': # Si nous avons le pays de la protéine coura
nte
            idx_country = list(country_latlon['country']).index(country[i]) #nou
s cherchons ses longitudes et latitudes
            current_lat_lon['lat'] = country_latlon['latitude'][idx_country]
            current_lat_lon['lon'] = country_latlon['longitude'][idx_country]
            lat_lon[i] = current_lat_lon
            name.append(protein_info[i]['GBSeq_accession-version']) #nous stocko
ns l'accession.version pour plus tard
```

→ Connaissant la latitude et la longitude de la majorité des isolats, représentez-les graphiquement sur une carte du monde:

In [12]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

# print(len(lat))

fig = plt.figure(num=None,figsize=(10,7),dpi=150) # ouvre une fenêtre
m = Basemap(projection='kav7',lon_0=0,lat_0=60) # carte du monde avec projection
for current_lat_lon in lat_lon: # On parcourt chaqunes des latitudes et longitud
es enregistrée
    x, y = m(current_lat_lon['lon'],current_lat_lon['lat']) # transforme lat,lon
avec la projection utilisée
    m.scatter(x,y,7,marker='.',color='r') # représente les isolats en rouge
m.fillcontinents(zorder=0) # dessine les continents
plt.show()
fig.savefig('map_isolats.png', dpi=fig.dpi) # sauve la figure
```

1.6 A votre avis, s'agit-il d'un virus qui se reproduit dans les eaux chaudes ou plutôt dans les eaux froides?

Plutôt dans les eaux froides, À l'intérieur des terres, il y a plus de chance que les points d'eaux soient en hauteur

2. Alignement des séquences protéiques pour pouvoir les comparer

Afin de comprendre comment le virus s'est propagé dans le monde, nous allons comparer les séquences protéiques des différents isolats. Ceci nous permettra de regrouper les isolats "les plus similaires" et nous ferons l'hypothèse vraisemblable qu'il ont une origine commune du fait de leur similarité.

Pour comparer les séquences protéiques et voir à quel point elles diffèrent, il est nécessaire de commencer par les aligner. Vous allez apprendre à utiliser l'outil graphique Aliview pour aligner les séquences et les inspecter visuellement. Vous verrez par la suite comment réaliser ces mêmes opérations directement en python avec la librairie biopython, ce qui vous permettra d'automatiser le processus.

→ **Utilisez la liste de gi pour pouvoir générer un fichier fasta contenant toutes les séquences protéiques qui nous intéressent (pour rappel, notre requête initiale était: `'G[gene] AND VHSV[orgn] NOT "mRNA"[title]"`**

In [13]:
```
from Bio import SeqIO
print(len(gi))
#print(len(giWCountry))
protein_seq = Entrez.efetch(db="nucleotide",id=gi,rettype="fasta_cds_aa",retmode
="text")
#protein_seq = Entrez.efetch(db="nucleotide",id=giWCountry,rettype="fasta_cds_aa
",retmode="text")
protein_seq = list(SeqIO.parse(protein_seq, "fasta"))
print(len(protein_seq))
```

```
390
410
```

→ Imprimez l'élément 0 et l'élément 212 de `protein_seq`

In [14]:
```
#pas dans le labo juste pour moi
"""
for i in range(len(protein_seq)):
    print('--> '+str(i))
    print(protein_seq[i])
    print('')
    #pas besoin de cette cellule dans le questionnaire des étudiants 194
"""
```

Out[14]:
```
" \nfor i in range(len(protein_seq)):\n    print('--> '+str(i))\n    print(pro
tein_seq[i])\n    print('')\n    #pas besoin de cette cellule dans le question
naire des étudiants 194\n"
```

In [15]:
```
print(protein_seq[0])
print('')
#print(protein_seq[192])
#print('')
#print(protein_seq[193])
#print('')
print(protein_seq[212])
print('')
#print(protein_seq[195])
#print('')
print(len(protein_seq))
```

```
ID: lcl|MH836523.1_prot_AZH81386.1_1
Name: lcl|MH836523.1_prot_AZH81386.1_1
Description: lcl|MH836523.1_prot_AZH81386.1_1 [gene=G] [protein=glycoprotein]
[protein_id=AZH81386.1] [location=1..1524] [gbkey=CDS]
Number of features: 0
Seq('MEWNTFFLVILIIIIKSTTSQITQRPPVENISTYHADWDTPLYTHPSNCRKNSF...QMV', SingleLett
erAlphabet())

ID: lcl|KM244768.1_prot_AKC42401.1_3
Name: lcl|KM244768.1_prot_AKC42401.1_3
Description: lcl|KM244768.1_prot_AKC42401.1_3 [gene=M] [protein=matrix protein
] [protein_id=AKC42401.1] [location=2217..2822] [gbkey=CDS]
Number of features: 0
Seq('MTLFKRKRTILVPPPHITSNDEDRVSTILTEGTLTITGPPPGNQVDKICMAMKL...QPR', SingleLett
erAlphabet())

410
```

2.1 Que constatez-vous concernant le nom du gène de `protein_seq[212]` et le nombre d'entrées retournées? Pourquoi ce gène apparaît-il dans notre liste si la requête initiale exigeait "G[gene]" (astuce: allez voir sur le site de Genbank l'entrée correspondant à l'identifiant KM244768).

Le nom du gène est M et non G comme demandé dans la requête. Dans une de ses features, le gene G apparait:

```
gene    2911..4434
        /gene="G"
```

source (https://www.ncbi.nlm.nih.gov/nuccore/KM244768)

Avant de continuer, nous devons donc nous assurer que seules les séquences protéiques correspondant au gène G sont gardées.

→ **Faites une boucle pour tester le nom du gène**

In [16]:
```
print(len(protein_seq))
updated_protein_seq = [ ps for i,ps in enumerate(protein_seq) if ps.description.
find('gene=G')>0 ]
print(len(updated_protein_seq))

# verify visually that only G genes are left
desc = [ps.description for i,ps in enumerate(updated_protein_seq)]
desc
```

```
410
390
```

Out[16]:
```
['lcl|MH836523.1_prot_AZH81386.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AZH81386.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MH836522.1_prot_AZH81385.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AZH81385.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MF176930.1_prot_ATO91455.1_1 [gene=G] [protein=glycoprotein] [protein_id
=ATO91455.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MF176929.1_prot_ATO91454.1_1 [gene=G] [protein=glycoprotein] [protein_id
=ATO91454.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MF176928.1_prot_ATO91453.1_1 [gene=G] [protein=glycoprotein] [protein_id
=ATO91453.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MF176927.1_prot_ATO92013.1_1 [gene=G] [protein=glycoprotein] [protein_id
=ATO92013.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MF176926.1_prot_ATO92012.1_1 [gene=G] [protein=glycoprotein] [protein_id
=ATO92012.1] [location=1..1524] [gbkey=CDS]',
 'lcl|MF176925.1_prot_ATO92011.1_1 [gene=G] [protein=glycoprotein] [protein_id
=ATO92011.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728262.1_prot_AQT19224.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19224.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728261.1_prot_AQT19223.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19223.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728260.1_prot_AQT19222.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19222.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728259.1_prot_AQT19221.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19221.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728258.1_prot_AQT19220.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19220.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728257.1_prot_AQT19219.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19219.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728256.1_prot_AQT19218.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19218.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728255.1_prot_AQT19217.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19217.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728254.1_prot_AQT19216.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19216.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728253.1_prot_AQT19215.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19215.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728252.1_prot_AQT19214.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19214.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728251.1_prot_AQT19213.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19213.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728250.1_prot_AQT19212.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19212.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728249.1_prot_AQT19211.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19211.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728248.1_prot_AQT19210.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19210.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728247.1_prot_AQT19209.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19209.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728246.1_prot_AQT19208.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19208.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728245.1_prot_AQT19207.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19207.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728244.1_prot_AQT19206.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19206.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728243.1_prot_AQT19205.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19205.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728242.1_prot_AQT19204.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19204.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728241.1_prot_AQT19203.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19203.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728240.1_prot_AQT19202.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19202.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728239.1_prot_AQT19201.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19201.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728238.1_prot_AQT19200.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19200.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728237.1_prot_AQT19199.1_1 [gene=G] [protein=glycoprotein] [protein_id
=AQT19199.1] [location=1..1524] [gbkey=CDS]',
 'lcl|KU728236.1_prot_AQT19198.1_1 [gene=G] [protein=glycoprotein] [protein_id
```

→ Maintenant que vous avez les séquences du gène G uniquement, remplacez l'id par le numéro d'accès et le nom du pays source (ceci facilitera les analyses par la suite).

In [17]:
```
print(len(updated_protein_seq))
for i,ps in enumerate(updated_protein_seq):
    access_number = ps.name.split('|')[1].split('_')[0]
    #print(ps) del liste[1]
    #if(country[i]):
    print(str(i)+" "+country[i])
    country_tmp = country[i].replace(' ','-')
    #print(country_tmp)
    updated_protein_seq[i].description = ''
    updated_protein_seq[i].id = access_number+'_'+country_tmp
```

```
390
0 Iceland
1 Iceland
2 Finland
3 Finland
4 Finland
5 Finland
6 Finland
7 Finland
8 Denmark
9 Denmark
10 Denmark
11 Denmark
12 Denmark
13 Denmark
14 Denmark
15 Denmark
16 Denmark
17 Denmark
18 Denmark
19 Denmark
20 Denmark
21 Denmark
22 Denmark
23 Denmark
24 Denmark
25 Denmark
26 Denmark
27 Denmark
28 Denmark
29 Denmark
30 Denmark
31 Denmark
32 Denmark
33 Denmark
34 Denmark
35 Denmark
36 Denmark
37 Denmark
38 Denmark
39 Denmark
40 Denmark
41 Denmark
42 Denmark
43 Denmark
44 Denmark
45 Denmark
46 Denmark
47 Denmark
48 Denmark
49 Denmark
50 Denmark
51 Denmark
52 Denmark
53 Denmark
54 Denmark
55 Denmark
56 Denmark
57 Denmark
58 Denmark
59 Denmark
60 Denmark
61 Denmark
62 Denmark
63 Denmark
64 Denmark
65 Denmark
66 Denmark
67 Denmark
```

→ Créez un fichier FASTA avec ces séquences et ouvrez-le avec un éditeur de texte pour voir ce qu'il contient.

In [18]:
```
output_handle = open('labo-2_protein-sequences.fasta', 'w')
SeqIO.write(updated_protein_seq, output_handle, "fasta")
output_handle.close()
```

Instalation de Aliview: https://ormbunkar.se/aliview/ (https://ormbunkar.se/aliview/)

→ Sur votre ordinateur, lancez le programme Aliview et utilisez File > Open File pour ouvrir le fichier que vous venez de créer.

→ Sur Aliview, allez sur Align > Realign everything et cliquez sur Ok.

Par défaut, Aliview utilise l'algorithme MUSCLE pour aligner les séquences.

→ Sur Aliview, inspectez l'alignement visuellement et assurez-vous que tout semble normal.

2.2 Pourquoi est-ce que l'une des séquences est plus courte que les autres? Astuce: allez sur Genbank depuis votre navigateur pour voir l'entrée correspondant à la séquence très courte.

Ce n'est pas la séquence complète

Viral hemorrhagic septicemia virus isolate KV010308-4 nucleoprotein (N), phosphoprotein (P), and matrix protein (M) genes, complete cds; and glycoprotein (G) gene, **partial cds**.

FJ362514.1 (https://www.ncbi.nlm.nih.gov/nuccore/FJ362514.1)

What is the mean of Cds, when it say in a sequence "partial Cds"? (https://www.researchgate.net/post/What_is_the_mean_of_Cds_when_it_say_in_a_sequence_partial_Cds)

Sur Aliview, il est possible d'enregistrer l'alignement: File > Save as Clustal (aln).

Pour la suite du labo, nous allons générer ce même alignement à l'aide de la librairie biopython et faire quelques analyses sur cet alignement.

→ Importez les modules suivants:

In [19]:
```
from Bio.Align.Applications import MuscleCommandline
from Bio import AlignIO
```

→ Spécifiez le chemin vers votre exécutable muscle

faute que j'installe muscle sur mon ordi...

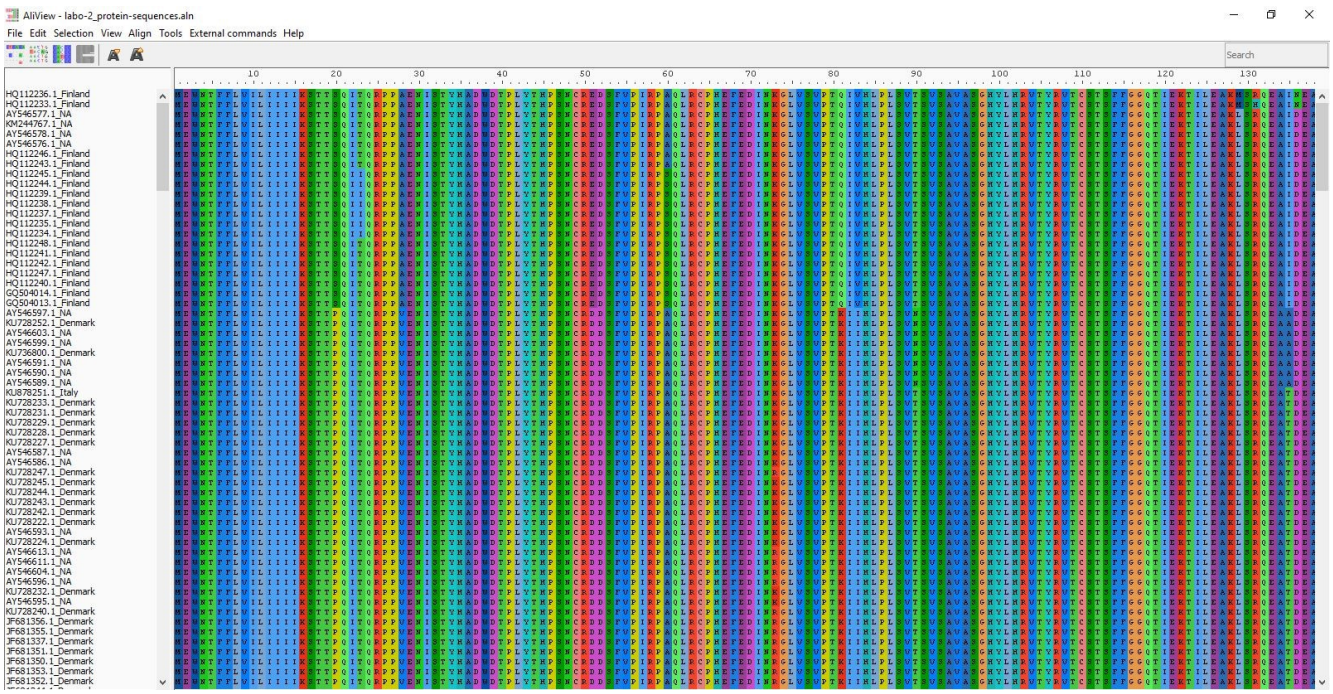
https://drive5.com/muscle/downloads.htm (https://drive5.com/muscle/downloads.htm)

In [20]:
```
#muscle_loc = r'/usr/bin/muscle' # modifier si nécessaire
#muscle_loc = r'/Users/xavierbrochet/Documents/programmes/muscle3.8.31_i86darwin64'
muscle_loc = r'muscle3.8.31_i86win32'
```

→ Réalisez l'alignement à partir du fichier FASTA contenant les séquences

In [21]:
```
in_file = 'labo-2_protein-sequences.fasta'
out_file = 'labo-2_protein-sequences.aln'

muscle_cline = MuscleCommandline(cmd=muscle_loc,input=in_file,out=out_file,clwst
rict=True)
#print(muscle_cline)
stdout, stderr = muscle_cline()

muscle_align = AlignIO.read(out_file,'clustal') # this command actually performs
the alignment
print(muscle_align)
```

```
SingleLetterAlphabet() alignment with 390 rows and 507 columns
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112236.1_Finland
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112233.1_Finland
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI AY546577.1_NA
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI KM244767.1_NA
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI AY546578.1_NA
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI AY546576.1_NA
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112246.1_Finland
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112243.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112245.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112244.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112239.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112238.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112237.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112235.1_Finland
MEWNTFFLVILIIIIKSTTSQIIQRPPAENISTYHADWDTPLYT...QMI HQ112234.1_Finland
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112248.1_Finland
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112241.1_Finland
MEWNTFFLVILIIIIKSTTSQITQRPPAENISTYHADWDTPLYT...QMI HQ112242.1_Finland
...
MEWNTFFLVILIIIIKSTTSQITQRPPVENISTYHADWDTPLYT...QMV DQ401188.1_Canada
```



Le fichier .aln généré peut être ouvert dans Aliview pour l'inspecter visuellement.

3. Construction de l'arbre phylogénétique et clustering hiérarchique pour identifier des groupes similaires

A partir de cet alignement, il est possible de générer un arbre phylogénétique. Par souci de temps, nous utiliserons FastTree qui est très rapide, même si cet outil n'est pas le mieux adapté pour construire un arbre phylogénétique à partir de séquences très proches comme c'est le cas ici.

→ **Générez l'arbre avec FastTree** INSTALLTION: http://microbesonline.org/fasttree/#Install (http://microbesonline.org/fasttree/#Install)

In [22]:
```
from Bio.Phylo.Applications import FastTreeCommandline

# convert aln to phy format
out_phy = 'labo-2_protein-sequences.phy'
AlignIO.convert(out_file,'clustal',out_phy, 'phylip-relaxed')

# generate the phylogenetic tree with FastTree
out_tree = 'labo-2_protein-sequences.tre'
#cmd_fasttree = r'/home/aitana/Applications/FastTree'
#cmd_fasttree = r'/Users/xavierbrochet/Documents/programmes/FastTree' # modifier si nécessaire
cmd_fasttree = r'FastTree' # modifier si nécessaire

fasttree_cmdline = FastTreeCommandline(cmd=cmd_fasttree,fastest=True,input=out_phy,out=out_tree)
print(fasttree_cmdline)
out_log, err_log = fasttree_cmdline()

# load tree
from Bio import Phylo
tree = Phylo.read('labo-2_protein-sequences.tre', 'newick')
#print(tree)
#Phylo.draw(tree)
```

```
FastTree -fastest -out labo-2_protein-sequences.tre labo-2_protein-sequences.phy
```

→ **Ouvrez le fichier `labo-2_protein-sequences.tre` dans TreeViewX et choisissez le mode de visualisation "phylogram" (bouton en haut à droite) et zoomez pour voir les groupes.**

Utilisation de http://ab.inf.uni-tuebingen.de/data/software/dendroscope3/download/welcome.html (http://ab.inf.uni-tuebingen.de/data/software/dendroscope3/download/welcome.html) ou de iToL en ligne... https://itol.embl.de/ (https://itol.embl.de/)

3.1 De quels pays vient l'isolat le plus proches des isolats de Corée du Sud?

Du japon

Combien de groupes voyez-vous? Ca dépend! Le nombre de groupes dépend du seuil utilisé pour admettre, ou non, que deux isolats font partie du même groupe.

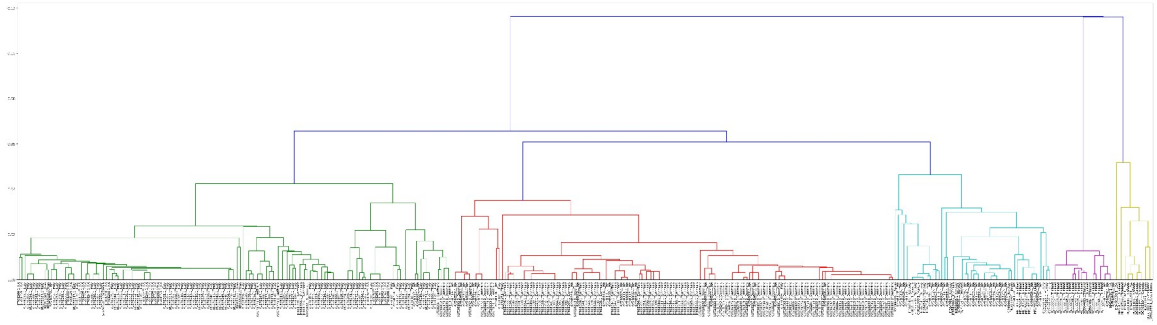
Afin de mieux visualiser la propagation du virus dans le monde, nous allons regrouper les séquences (i.e. les isolats) en 4 groupes et voir leur localisation géographique sur une carte du monde. Pour obtenir ces 4 groupes, nous allons commencer par calculer les distances (phylogénétiques) entre les isolats et réaliser un clustering hiérarchique à partir duquel nous obtiendrons 4 groupes.

→ **Calculez les distances entre les isolats** (ceci peut prendre un petit moment...)

In [23]:
```
dmat = []
leaves = [str(cladit) for k,cladit in enumerate(tree.get_terminals())]
for l1,leave1 in enumerate(leaves):
    d = []
    for l2,leave2 in enumerate(leaves):
        d.append(tree.distance(leave1,leave2))
    dmat.append(d)
```

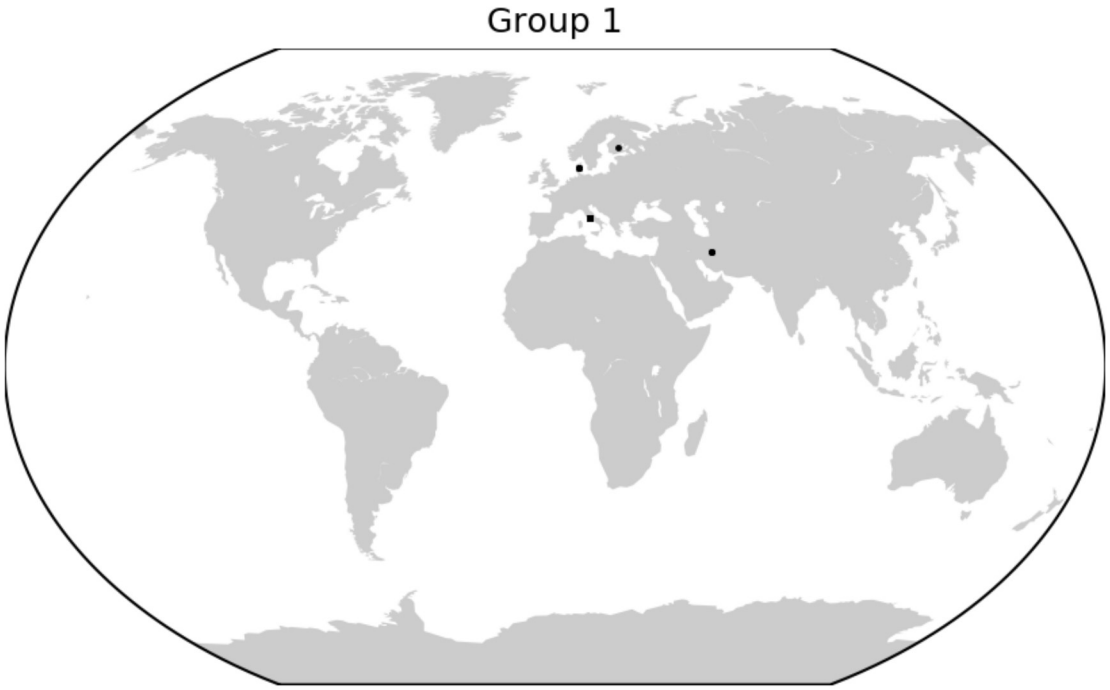
→ **Réalisez le clustering hiérarchique et visualisez le dendrogram obtenu (double-cliquez sur le graphique pour l'agrandir)**

In [24]:
```
import scipy.cluster.hierarchy as cl
import scipy.spatial.distance as ssd

Z = cl.linkage(ssd.squareform(dmat),method='average',metric='euclidean')
fig = plt.figure(num=None,figsize=(60,15),dpi=250)
dendro=cl.dendrogram(Z,labels=leaves,color_threshold=0.06,leaf_rotation=90,leaf_
font_size=9)
plt.show()
```

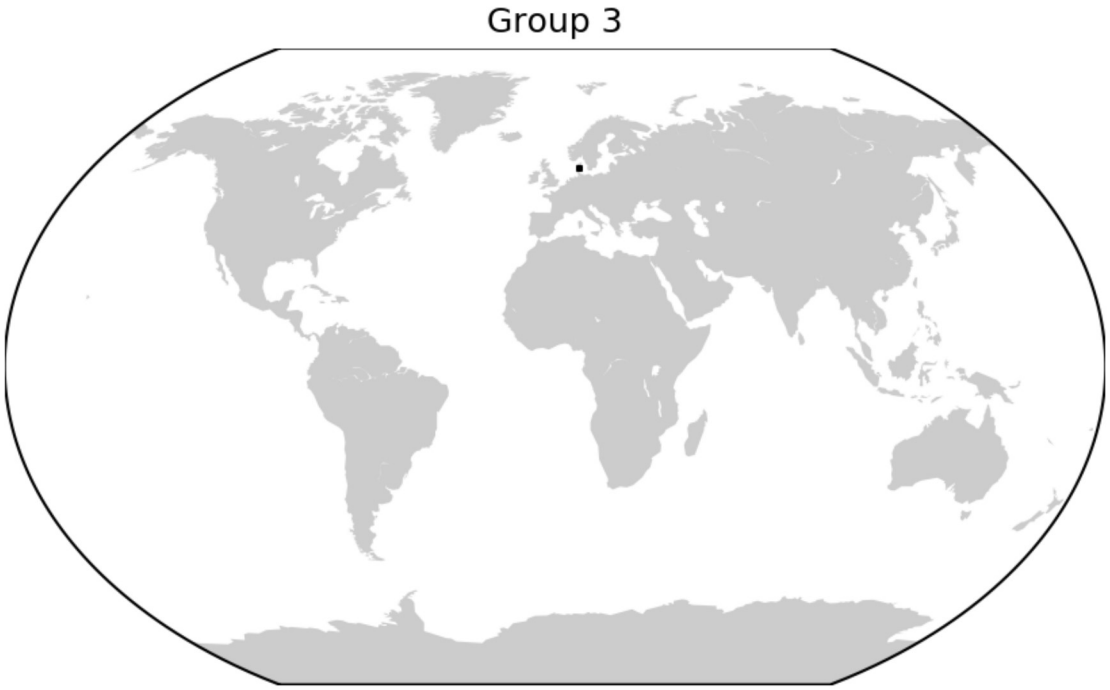


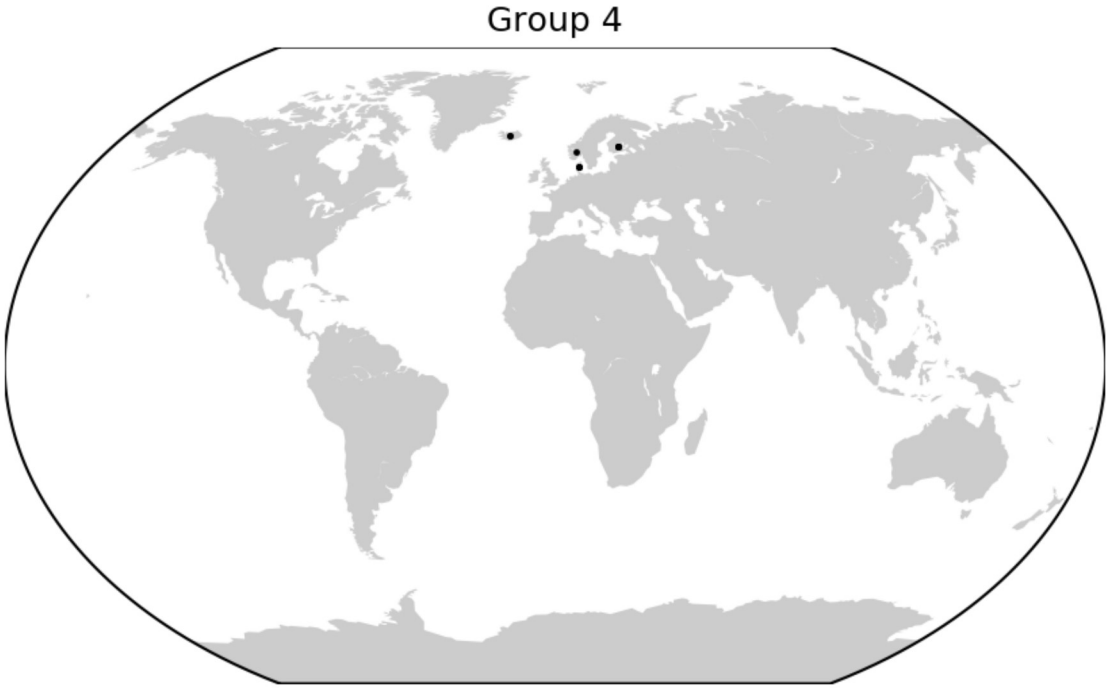
→ Représentez les isolats de chaque groupe sur une carte du monde

In [25]:
```
nb_clusters = 4
clusters = cl.fcluster(Z,nb_clusters,criterion='maxclust')

lat = []
lon = []

for i,current_lat_lon in zip(range(len(lat_lon)),lat_lon):
    lat.append(current_lat_lon['lat'])
    lon.append(current_lat_lon['lon'])

for i in range(1,max(clusters)+1):

    idxi = [j for j,cluster in enumerate(clusters) if cluster==i]

    lat_tmp = []
    lon_tmp = []
    for j,idxj in enumerate(idxi):
        access_number_of_leave = leaves[idxj].split('_')[0]
        if access_number_of_leave in name:
            index = name.index(access_number_of_leave)
            lat_tmp.append(lat[index])
            lon_tmp.append(lon[index])

        #lat_tmp = [lat[name.index(access_number_of_leave)] for ]
        #lon_tmp = [lon[name.index(access_number_of_leave)] for j,idxj in enumer
ate(idxi)]

    # plot
    fig = plt.figure(num=None,figsize=(7,6),dpi=150)
    m = Basemap(projection='kav7',lon_0=0,lat_0=60)
    x, y = m(lon_tmp,lat_tmp) # transform (lat,lon) with the projection
    m.scatter(x,y,7,marker='.',color='k')
    m.fillcontinents(zorder=0)
    plt.title('Group '+str(i))
    plt.show()
    fig.savefig('map_isolats-'+str(i)+'.png', dpi=fig.dpi)
```


Group 1


Group 2


Group 3


Group 4

Sur plusieurs de ces figures, vous pouvez voir que le virus “voyage” parfois même de très longues distances. L’une des raisons principales de cette propagation vient de l’échange d’oeufs et de larves de poissons contaminés entre pisciculteurs du monde entier (p.ex. entre l’Amérique du Nord et l’Asie).

3.2 A l’aide du tableau ci-dessous, définissez le(s) genotype(s) probable auxquels appartiennent les 4 groupes identifiés.

Nota: Ces génotypes ont été définis en comparant les séquences des protéines G et N, alors qu’ici nous n’avons étudié que la protéine G.

Type	Prevalent host type and location (source: Wikipedia (https://en.wikipedia.org/wiki/Viral_hemorrhagic_septicemia))
I-a	Farmed rainbow trout and a few other freshwater fish in continental Europe
I-b	Marine fish of the Baltic Sea, Skagerrak, Kattegat, North Sea, Japan
I-c	Farmed rainbow trout Denmark
I-d	Farmed rainbow trout in Norway, Finland, Gulf of Bothnia
I-e	Rainbow trout in Georgia, farmed and wild turbot in the Black Sea
II	Marine fish of the Baltic Sea
III	Marine fish of the British Isles and northern France, farmed turbot in the UK and Ireland, and Greenland halibut (Reinhardtius hippoglossoides) in Greenland
IV-a	Marine fish of the Northwest Pacific (North America), North American north Atlantic coast, Japan, and Korea
IV-b	Freshwater fish in North American Great Lakes region

Tout dépend de l'interprétation que l'on en fait, il faudrait appliquer un modèle de logique floue, je n'ai pris que les type qui englobait tous les points d'un groupe, pour tous les types qui contenaient au moins un élément d'un groupe

Groupe	Type(s)
Groupe 1	?
Groupe 2	I-a, I-b
Groupe 3	I-b, I-c, II
Groupe 4	?

4. Pour aller plus loin… L’épidémiologie

Les virus évoluent très rapidement à l’échelle de la vie humaine, et peuvent se propager rapidement à travers le monde. Un exemple de virus qui évolue très rapidement est la grippe saisonnière. Après chaque hiver, une grande partie de la population développe des anticorps contre la grippe de cette année et est donc immunisée… Mais comme le virus de la grippe évolue rapidement, l’année suivante, un nouveau mutant apparaît pour lequel la majorité de la population n’est pas complètement immunisée. Le vaccin basé sur la grippe de l’année précédente ne protège donc que partiellement la population, et est totalement inefficace dans les rares cas où un nouveau sous-type de grippe apparaît, créant un grand risque de pandémie. La bioinformatique est très utile pour suivre l’évolution de la grippe et comprendre comment elle se propage.

4.1 Par rapport à l’analyse faite ici, quel facteur important faudrait-il prendre en compte pour pouvoir identifier les “routes” empruntées par le virus pour se propager (i.e. “de où à où il va”)? (50 mots maximum)

Un virus va prendre les routes commerciales, aller d'un point où un produit est récolté et produit pour voyager parmi tous les transports et les lieux de transformations ou d'utilisation(Les oeufs échangés entre différents pisciculteurs). Le tourisme est aussi compté dans les routes commerciales.

4.2 Le [virus Zika](https://www.who.int/wer/2015/wer9045.pdf) a fait la une de l'actualité il y a quelque temps (2015). En effet, au Brésil, les autorités sanitaires locales ont observé une recrudescence de cas atteints du syndrome Guillain-Barré qui coincident avec des cas d'infections à virus Zika dans le grand public, ainsi qu'une augmentation du nombre de nouveau-nés atteints de microcéphalie dans le nord-est du pays [source: site OMS]. Répétez l'analyse de la Section 1 avec la requête `'zika virus[orgn] NOT "mRNA"[title]'`. Est-ce que le virus a été isolé et séquencé en Argentine (si oui, combien de fois)?

Commencez par récupérer les identifiants gi correspondant à la requête en utilisant Entrez.esearch()

In [26]:
```
query = 'zika virus[orgn] NOT "mRNA"[title]'
handle = Entrez.esearch(db='nucleotide',retmax=1000,term=query)
record = Entrez.read(handle)
gi = record['IdList']
#print(gi)
print(len(gi))
```

```
1000
```

Allez chercher les informations Genbank sur ces séquences en utilisant Entrez.efetch()

In [27]:
```
protein_info = Entrez.efetch(db="nucleotide",id=gi,rettype="gb",retmode="xml")
protein_info = Entrez.read(protein_info)
print(len(protein_info))
```

```
1000
```

Généralisez ce code pour qu'il imprime le pays d'origine des 1000 isolats et enregistre le résultat dans une variable (liste de 1000 éléments) appelée country (si le pays n'est pas disponible, ajoutez le texte 'NA' (not available) à la place).

In [28]:
```
country = []

for info in protein_info:
    current_country = 'NA'
    for j,feat in enumerate(info['GBSeq_feature-table'][0]['GBFeature_quals']):

        if feat['GBQualifier_name']=='country':
            current_country = feat['GBQualifier_value'].split(':')[0]
    country.append(current_country)

print(len(country))
```

```
1000
```

Généralisez votre code pour qu'il enregistre aussi la latitude et la longitude dans deux variables séparées (mettez un float('nan') si elles ne sont pas disponibles). Astuce: Utilisez la fonction split(' ') pour extraire les nombres qui vous intéressent de lat_lon (à convertir en float aussi).

In [29]:
```
lat_lon = []

for info in protein_info:
    current_lat_lon = {'lat': float('nan'), 'lon': float('nan')}
    for j,feat in enumerate(info['GBSeq_feature-table'][0]['GBFeature_quals']):

        if feat['GBQualifier_name']=='lat_lon': #J'ai essayé de prendre en compt
e les directions W et S, mais sans tester + mon code
            current = feat['GBQualifier_value'].split('N')
            if len(current) < 2:
                current = feat['GBQualifier_value'].split('S')
                lat = '-{}'.format(current[0])
            else:
                lat = current[0]

            current_lat_lon['lat'] = lat

            current = current[1].split('E')
            if len(current) < 2:
                current = current[0].split('W')
                lon = '-{}'.format(current[0])
            else:
                lon = current[0]

            current_lat_lon['lon'] = lon
    lat_lon.append(current_lat_lon)

print(len(lat_lon))
```

```
1000
```

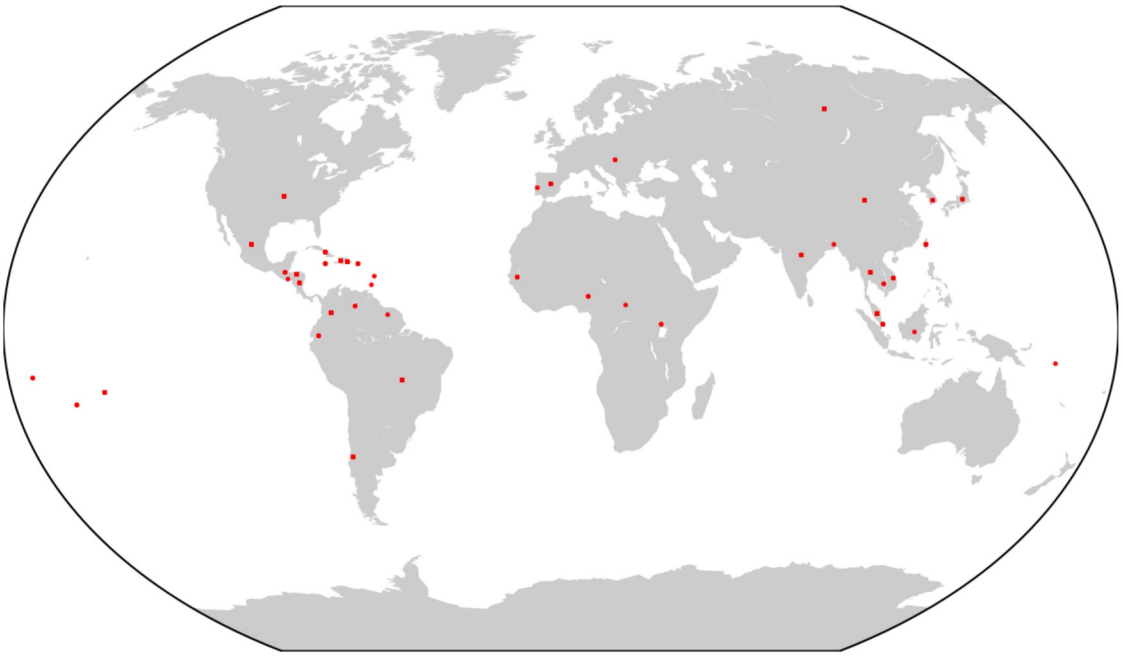
Généralisez votre code pour qu'il enregistre la latitude et la longitude moyenne lorsque celles-ci ne sont pas disponibles mais que vous connaissez le pays d'origine.

In [30]:
```
for i in range(len(lat_lon)):
    current_lat_lon = {'lat': float('nan'), 'lon': float('nan')}
    if lat_lon[i]['lat'] != None:
        current_country = country[i]
        if current_country != 'NA':
            idx_country = list(country_latlon['country']).index(country[i])
            current_lat_lon['lat'] = country_latlon['latitude'][idx_country]
            current_lat_lon['lon'] = country_latlon['longitude'][idx_country]
            lat_lon[i] = current_lat_lon
print(len(lat_lon))
```

```
1000
```

Connaissant la latitude et la longitude de la majorité des isolats, représentez-les graphiquement sur une carte du monde:

In [31]:
```
fig = plt.figure(num=None,figsize=(10,7),dpi=150) # ouvre une fenêtre
m = Basemap(projection='kav7',lon_0=0,lat_0=60) # carte du monde avec projection
for current_lat_lon in lat_lon: # On parcourt chaqunes des latitudes et longitud
es enregistrée
    x, y = m(current_lat_lon['lon'],current_lat_lon['lat']) # transforme lat,lon
avec la projection utilisée
    m.scatter(x,y,7,marker='.',color='r') # représente les isolats en rouge
m.fillcontinents(zorder=0) # dessine les continents
plt.show()
fig.savefig('map_isolats.png', dpi=fig.dpi) # sauve la figure
```



Nous pouvons voir sur cette carte que le virus n'a pas été isolé qu'à l'Argentine, mais qu'il a fait le tour du monde.

In [32]:
```
country.count('Argentina')
```

Out[32]:
```
0
```

Aucune séquence n'a été trouvé en Argentine pour cette base de données