Bio.PopGen: Populyasiya genetikası Filogenetik ağac

GenePop nədir?

GenePop – populyasiya genetikası sahəsində geniş istifadə olunan proqram paketidir. Aşağıdakı analizləri dəstəkləyir:

- **Hardy-Weinberg balansı testləri**
- **Bağlılıq disbalansı (linkage disequilibrium)**
- **Populyasiyalar arasında fərqlilik (differentiation)**
- **Əsas statistikalar**
- **Fst və miqrasiya qiymətləndirmələri**

GenePop **ardıcıllıq əsaslı (sequence-based)** statistikaları dəstəkləmir, çünki **ardıcıllıq məlumatları ilə işləməyə** uyğun deyil.

Biopython və GenePop

Biopython daxilindəki **Bio.PopGen** modulu:

•**GenePop fayllarını oxumaq və yaratmaq** imkanı verir,

•Fayl məzmunu üzərində dəyişiklik etmək üçün **alətlər (utilitilər)** təqdim edir.

```
from Bio.PopGen import GenePop

with open("example.gen") as handle:
    rec = GenePop.read(handle)
```

Bu kod example.gen adlı GenePop faylını oxuyur və rec adlı dəyişənə yükləyir.

```
print(rec)
```

Əgər print(rec) etsəniz, məlumat GenePop formatında yenidən çap olunacaq.

Əsas məlumatlar rec dəyişənində:

•Lokus (genetik marker) adları:

```
rec.loci_list
```

Populyasiya məlumatları:

```
rec.populations
```

rec.populations strukturu necə işləyir?

rec.populations – hər biri bir populyasiyanı təmsil edən siyahıdır.
Hər bir populyasiya – fərdlər siyahısıdır, hər fərd isə bir **ad** və **allel cütlərindən** ibarətdir (hər lokus üçün 2 allel).

```
[
    [
        ("Ind1", [(1, 2), (3, 3), (200, 201)]),
        ("Ind2", [(2, None), (3, 3), (None, None)]),
    ],
    [
        ("Other1", [(1, 1), (4, 3), (200, 200)]),
    ],
]
```

Bu nümunədə:

•**İki populyasiya** var.

•**Birinci populyasiyada 2 fərd**:

- •Ind1 – 3 lokus üçün tam allel məlumatı var.
- •Ind2 – bəzi lokuslarda məlumat **çatışmır** (məsələn, (None, None)).

•**İkinci populyasiyada 1 fərd** – Other1.

❖ GenePop Qeydlərinin İdarə Olunması Üçün Faydalı Funksiyalar

Bio.PopGen.GenePop modulunda, **GenePop formatındakı məlumatlar üzərində əməliyyat aparmaq** üçün bir neçə **istifadəçi yönümlü funksiya** təqdim olunur.

```
from Bio.PopGen import GenePop

# Təsəvvür edin ki, daha əvvəl aşağıdakı kodla GenePop faylını oxumusunuz:
# with open("example.gen") as handle:
#     rec = GenePop.read(handle)
```

❖ rec.remove_population(pos)

- rec.populations siyahısında **istədiyiniz populyasiyanı silir**.
- pos – silmək istədiyiniz populyasiyanın **sıra nömrəsidir** (0-dan başlayır).
- Məsələn, birinci populyasiyanı silmək üçün rec.remove_population(0) istifadə olunur.
- Bu funksiya **mövcud rec obyektini dəyişir**.

rec.remove_locus_by_position(pos)

- rec.loci_list siyahısındakı **istədiyiniz lokusu (genetik marker) silir**.
- pos – silmək istədiyiniz lokusun **sıra nömrəsidir** (0-dan başlayır).
- Məsələn, ikinci lokusu silmək üçün rec.remove_locus_by_position(1).
- Bu da **rec obyektini dəyişdirir**.

rec.remove_locus_by_name(name)

- Lokusun **adına görə** onu silir.
- name – silmək istədiyiniz lokusun adı (rec.loci_list daxilindəki ad).
- Əgər adı tapmasa, **funksiya səssizcə heç bir əməliyyat etməz** (xəta çıxmaz).
- rec dəyişdirilir.

rec_loci = rec.split_in_loci()

•**Hər bir lokus üçün ayrıca bir GenePop obyekti** yaradır.

•Yəni hər rekordda yalnız **bir lokus və bütün populyasiyalar** olur.

•Nəticə **lokus adlarını açar (key) kimi istifadə edən sözlük (dictionary)** şəklində qaytarılır.

```
{
    "Lokus1": GenePop.Record,
    "Lokus2": GenePop.Record,
    ...
}
```

Əsas rec dəyişməz qalır.

rec_pops = rec.split_in_pops(pop_names)

•**Hər bir populyasiya üçün ayrıca GenePop obyekti** yaradır.

•Yəni hər rekordda yalnız **bir populyasiya və bütün lokuslar** olur.

•Nəticə **populyasiya adlarını açar (key) kimi istifadə edən sözlük** şəklində qaytarılır.

```
{
    "Pop1": GenePop.Record,
    "Pop2": GenePop.Record,
    ...
}
```

•**Diqqət**: GenePop formatında **populyasiya adları saxlanmır**, ona görə bu adları **pop_names siyahısı** kimi **əlavə ötürmək** lazımdır.

•rec dəyişməz qalır.

Phyloqenetik Ağaclarla İş — Bio.Phylo Modulu ilə

1. Faylın hazırlanması

Əvvəlcə simple.dnd adlı **Newick** formatında olan faylı yaradırıq. Bu fayl ağacın strukturu üçün məlumat saxlayır.

```
(((A,B),(C,D)),(E,F,G));
```

Bu struktur belədir:

- A və B bir qrupda
- C və D başqa bir qrupda
- Sonra bunlar birləşir
- E, F, G isə ayrı bir budaqda yerləşir

2. Python-da faylı oxuma

Python (və ya IPython) terminalına keçirik və əvvəlcə Bio modulunu çağırırıq və faylı oxuyur:

```
from Bio import Phylo
tree = Phylo.read("simple.dnd", "newick")
```

Sonra ağac obyektini çap edirik:

```
print(tree)
```

Çıxış aşağıdakı kimidir:

```
Tree(rooted=False, weight=1.0)
    Clade()
        Clade()
            Clade()
                Clade(name='A')
                Clade(name='B')
            Clade()
                Clade(name='C')
                Clade(name='D')
        Clade()
            Clade(name='E')
            Clade(name='F')
            Clade(name='G')
```

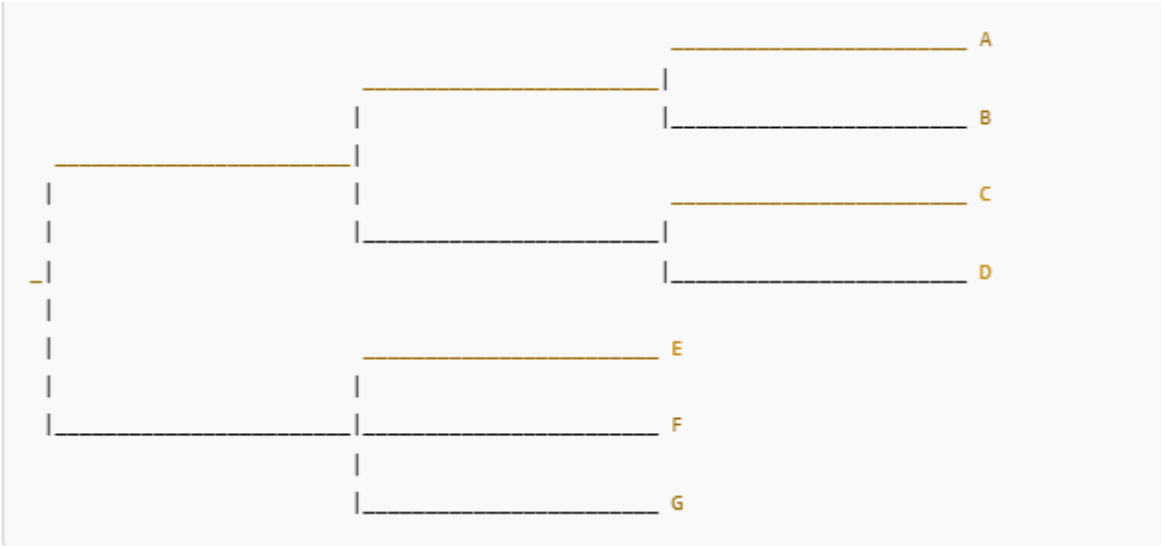
Bu, fylogenetik ağacın strukturunu tekst formatında göstərir.

3. ASCII formatında vizuallaşdırma

Bu addımda ağacı **ASCII** (mətn əsaslı) şəklində vizuallaşdıraq:

```
Phylo.draw_ascii(tree)
```

Nəticə belə olacaq:



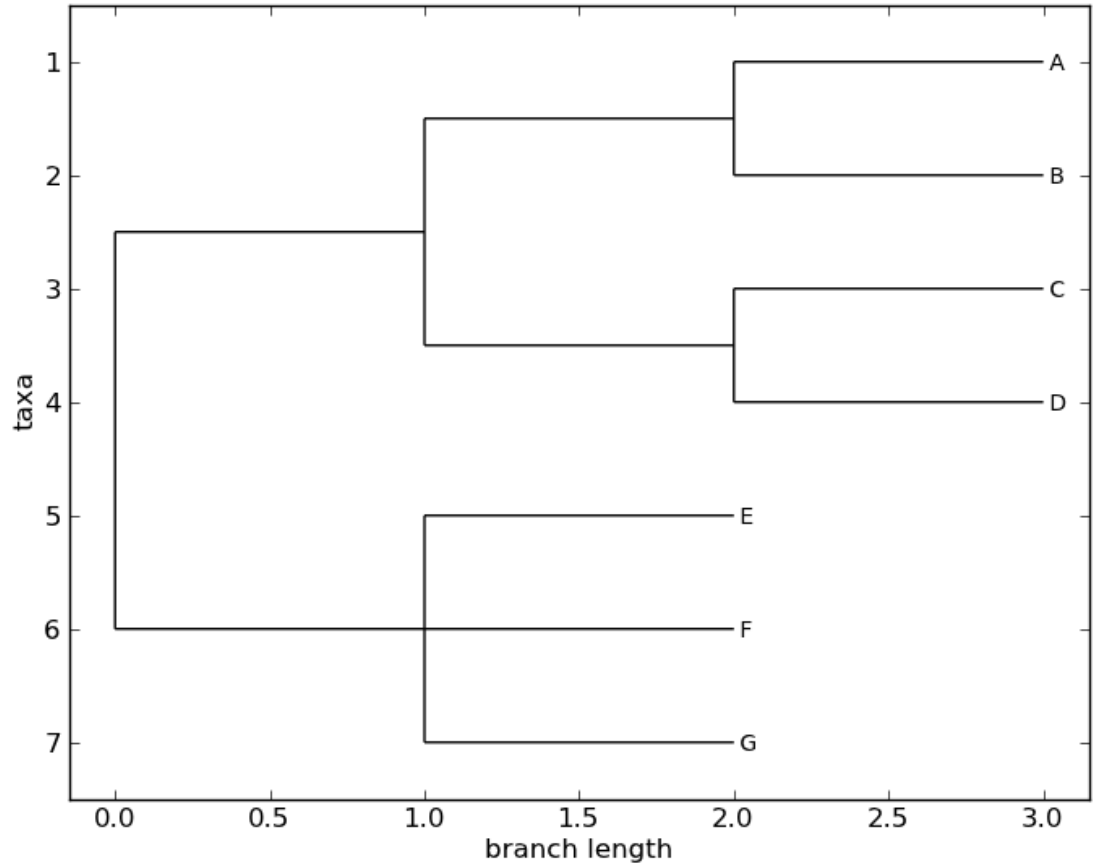
Bu, terminalda sadə şəkildə göstərilən fylogenetik ağacdır.

4. Qrafik vizuallaşdırma (matplotlib ilə)

Əgər matplotlib və ya pylab modulları qurulubsa, ağacı **qrafik** şəklində də çəkə bilərsiniz:

```
tree.rooted = True  # Ağacı köklü kimi qeyd edirik
Phylo.draw(tree)
```

Bu kod bizə qrafik pəncərədə budaqları ilə birlikdə gözəl bir fylogenetik ağac göstərəcək.



Phylogenetik Ağacın Budaqlarının Rənglənməsi

1. Yeni ağac yaradılır və PhyloXML formatına çevrilir:

Newick formatı rəngləmə və budaq qalınlığı kimi xüsusiyyətləri yadda saxlamır. Ona görə ağacı PhyloXML formatına çevirmək lazımdır:

```
from Bio import Phylo
from io import StringIO

# Sadə Newick ağacı
newick_tree = "(((A,B),(C,D)),(E,F,G));"
tree_handle = StringIO(newick_tree)
tree = Phylo.read(tree_handle, "newick")

# Rəngləmə üçün PhyloXML formatına çeviririk
tree = tree.as_phyloxml()
```

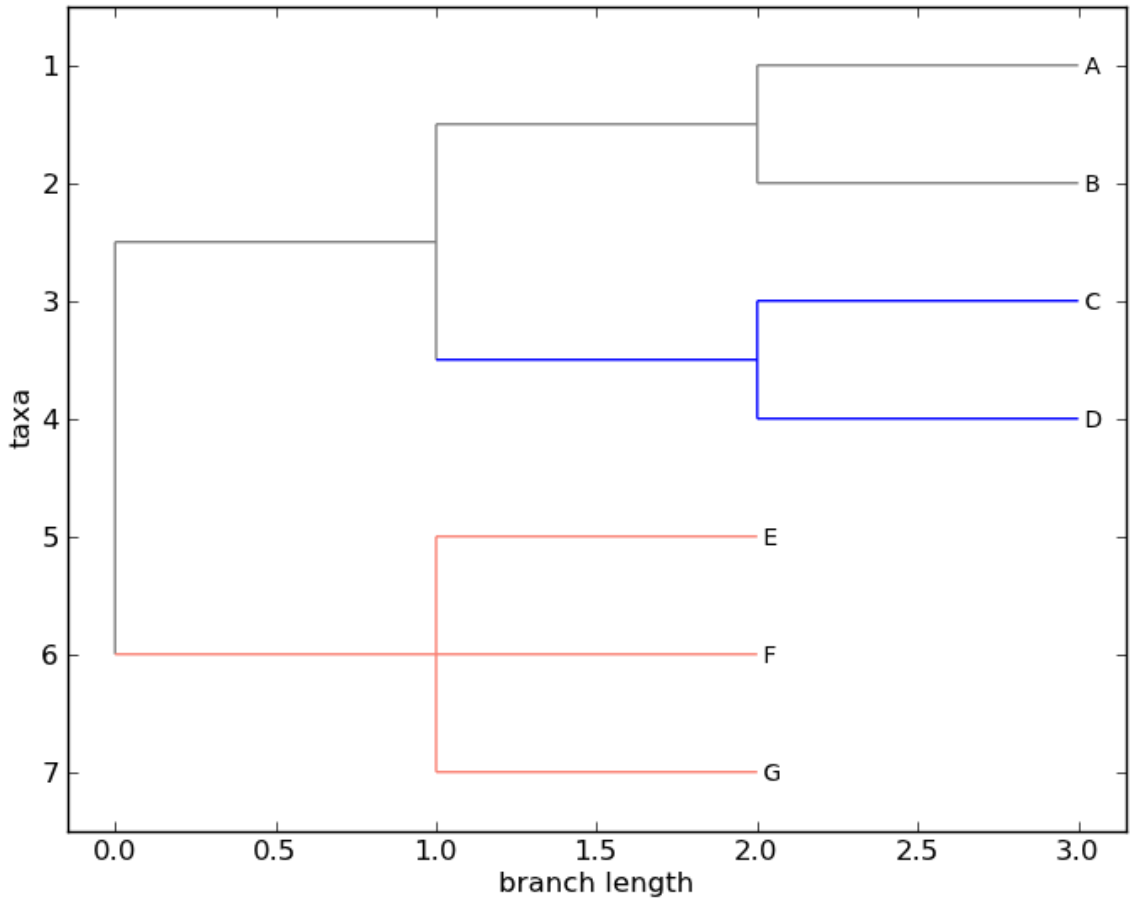
2. **Budaqlara rəng veririk**

```
# Kökü (root) boz rəngə boyayırıq
tree.root.color = "gray"

# "E" və "F" nöqtələrinin ortaq əcdadını tapıb "salmon" rənginə boyayırıq
mrca = tree.common_ancestor({"name": "E"}, {"name": "F"})
mrca.color = "salmon"

# Bəzi budaqlara birbaşa indekslə müdaxilə edərək rəng veririk
tree.clade[0, 1].color = "blue"  # Bu C və D olan hissədir
```

3. Rənglənmiş ağacı göstəririk

```
import matplotlib.pyplot as plt

# Vizual olaraq rəngli ağacı çəkirik
Phylo.draw(tree)
```

Bu kodları çalışdırdıqda biz boz köklü, salmon rəngli E-F hissəsi və mavi C-D budağı olan gözəl bir fylogenetik ağac görəcəyik.



Rənglənmiş Fylogenetik Ağacın Yadda Saxlanması və Fayl Əməliyyatları

1. Budaq rəngi nəyə təsir edir?

Bir **clade**-ə (Clade – budaq) verilən **rəng**, həmin budağa gələn budağı **və onun bütün alt hissələrini** əhatə edir.

Məsələn, “E” və “F” üçün ortaq əcdad mrca rənglənibsə, bu hissə aşağıya doğru "salmon" rəngində olacaq. Bu, **ağacın strukturu və harada kök olduğunu vizual şəkildə başa düşmək üçün faydalıdır.**

2. Rənglənmiş ağacı fayla necə yazırıq?

```
from Bio import Phylo
import sys

# Rənglənmiş ağacı phyloXML formatında ekrana (və ya fayla) yazırıq
n = Phylo.write(tree, sys.stdout, "phyloxml")
```

- sys.stdout istifadə olunduqda nəticə **ekranda** görünür.
- Phylo.write(tree, "mytree.xml", "phyloxml") istifadə etsəniz, nəticə **"mytree.xml"** adlı fayla yazılır.
- phyloxml formatı rəng, budaq qalınlığı və əlavə məlumatları **qoruyur**.
- Bu faylı **Archaeopteryx** kimi digər proqramlarla açanda rənglər görünəcək.

3. Fayl Əməliyyatları (Oxuma, Yazma, Dönüştürmə)

read() – Fayldan bir ağacı oxumaq:

```
from Bio import Phylo

tree = Phylo.read("Tests/Nexus/int_node_labels.nwk", "newick")
print(tree)
```

Diqqət: Faylda **yalnız bir ağac** varsa istifadə olunur. Əgər çoxdursa, **xəta verir**.

parse() – Bir neçə ağacı oxumaq (iterator kimi):

```
trees = Phylo.parse("Tests/PhyloXML/phyloxml_examples.xml", "phyloxml")
for tree in trees:
    print(tree)
```

Faylda **çoxlu ağaclar** olduqda istifadə olunur.

write() – Ağacı fayla yazmaq:

```
Phylo.write(tree1, "tree1.nwk", "newick")
```

və ya çoxlu ağacları:

```
python

Phylo.write(trees, "other_trees.xml", "phyloxml")
```

convert() – Formatlar arasında çevirmək:

```
Phylo.convert("tree1.nwk", "newick", "tree1.xml", "nexml")
Phylo.convert("other_trees.xml", "phyloxml", "other_trees.nex", "nexus")
```

StringIO – Fayl əvəzinə mətn ilə işləmək:

```
from io import StringIO

handle = StringIO("(((A,B),(C,D)),(E,F,G));")
tree = Phylo.read(handle, "newick")
```

Fayl yazmadan ağacı birbaşa **mətn formatında** istifadə etmək üçün çox faydalıdır.

Nəticə:

- PhyloXML formatı ilə rəngləri qoruyursunuz.
- read, write, parse, convert funksiyaları ilə ağaclarla rahat işləmək olur.
- StringIO sayəsində faylsız testlər etmək mümkündür.

Tree və Clade obyektləri ilə işləmək

Tree (ağac) obyektləri filogenetik ağacın əsas strukturudur. Bu obyektlər root adlı əsas (kök) Clade obyektini saxlayır.

•**Tree** — ümumi məlumatları (məsələn, ağacın köklü olub-olmaması kimi) saxlayır.

•**Clade** — hər bir düyün və alt ağac üçün spesifik məlumatları saxlayır (məsələn, budaq uzunluğu, alt Clade-lər və s.).

Həm Tree, həm də Clade obyektləri **TreeMixin** sinfindən miras alır, bu da onlara axtarış, araşdırma və dəyişiklik etməyə imkan verən bir çox metod verir.

Axtarış və Gəzinti Metodları

Əsas metodlar:

•get_terminals() — yalnız son (yarpaq) Clade-ləri (yəni, ucu) qaytarır.

•get_nonterminals() — yalnız daxili (aralıq) Clade-ləri qaytarır.

Əlavə metodlar:

•find_clades() — verilən şərtə uyğun Clade-ləri tapır.

•find_elements() — Clade-ə bağlı obyektləri tapır (PhyloXML fayllarında istifadə olunur).

•find_any() — ilk uyğun gələn obyekti tapır və ya None qaytarır.

•get_path(target) — kökdən target-ə qədər olan Clade-lərin siyahısını qaytarır.

•trace(start, finish) — başlanğıcdan son nöqtəyə qədər olan yolu verir.

Axtarış meyarları necə işləyir?

•Obyektlə (məsələn, bir Clade obyekti ilə).

•Ad ilə (məsələn, name="Foo1").

•Klass tipi ilə (məsələn, Clade).

•Sözlük ilə (məsələn, {"name": "Foo.*"} — regex dəstəklənir).

•Funksiya ilə (məsələn, lambda x: x.branch_length > 0.1).

Məlumatçı metodlar (ağac haqqında məlumat verir):

•common_ancestor() — verilən Clade-lərin ortaq əcdadını tapır.

•count_terminals() — yarpaq Clade-lərin sayını verir.

•depths() — hər Clade-in kökdən nə qədər uzaq olduğunu (dərinliyini) göstərən lüğət verir.

•distance(a, b) — iki Clade arasındakı ümumi budaq uzunluğunu qaytarır.

•total_branch_length() — ağacdakı bütün budaq uzunluqlarının cəmini verir.

Boolean (doğru/yanlış) metodlar:

•is_bifurcating() — ağac yalnız iki budaqlı düyünlərdən ibarətdirsə, True verir.

•is_monophyletic(targets) — Clade-lər eyni qrupdandırsa, True verir.

•is_parent_of(target) — verilən Clade, hazırkı Clade-in törəməsidirsə, True verir.

•is_preterminal() — əgər bütün törəmələr son Clade-dirsə, True.

Dəyişiklik metodları:

Bu metodlar ağacı yerində dəyişir. Əgər orijinal ağacı qorumaq istəyirsinizsə, əvvəlcə copy.deepcopy(tree) ilə kopyalayın.

•collapse(clade) — həmin Clade-i silir və uşaqlarını yuxarıya bağlayır.

•collapse_all() — yalnız son Clade-ləri saxlayır, qalanlarını silir.

•ladderize() — Clade-ləri terminalların sayına görə sıralayır.

•prune(clade) — bir yarpaq Clade-i ağacdan silir.

•root_with_outgroup(clades) — ağacı verilən Clade-lərə əsasən yenidən kökləyir.

•root_at_midpoint() — ağacı ən uzaq iki terminal arasında orta nöqtədən kökləyir.

•split(n=2) — Clade-i n hissəyə bölür.