```
In [60]:
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import seaborn as sns
from collections import OrderedDict

def get_gamma_from_file(file):
    lines = !cat {file}
    return float([i for i in lines if 'gamma' in i][0].split(' ')[-1])

def get_LL_from_file(filename):
    with open(filename) as f:
        lines = f.readlines()
    return float([i for i in lines if 'likelihood' in i and 'final' in i][0].split(' ')[-1])


plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams.update({'font.size': 25})
```

A tutorial on analysing parameter inference or decoding from cobraa.

Last updated 19th March 2024.

Inference of model parameters

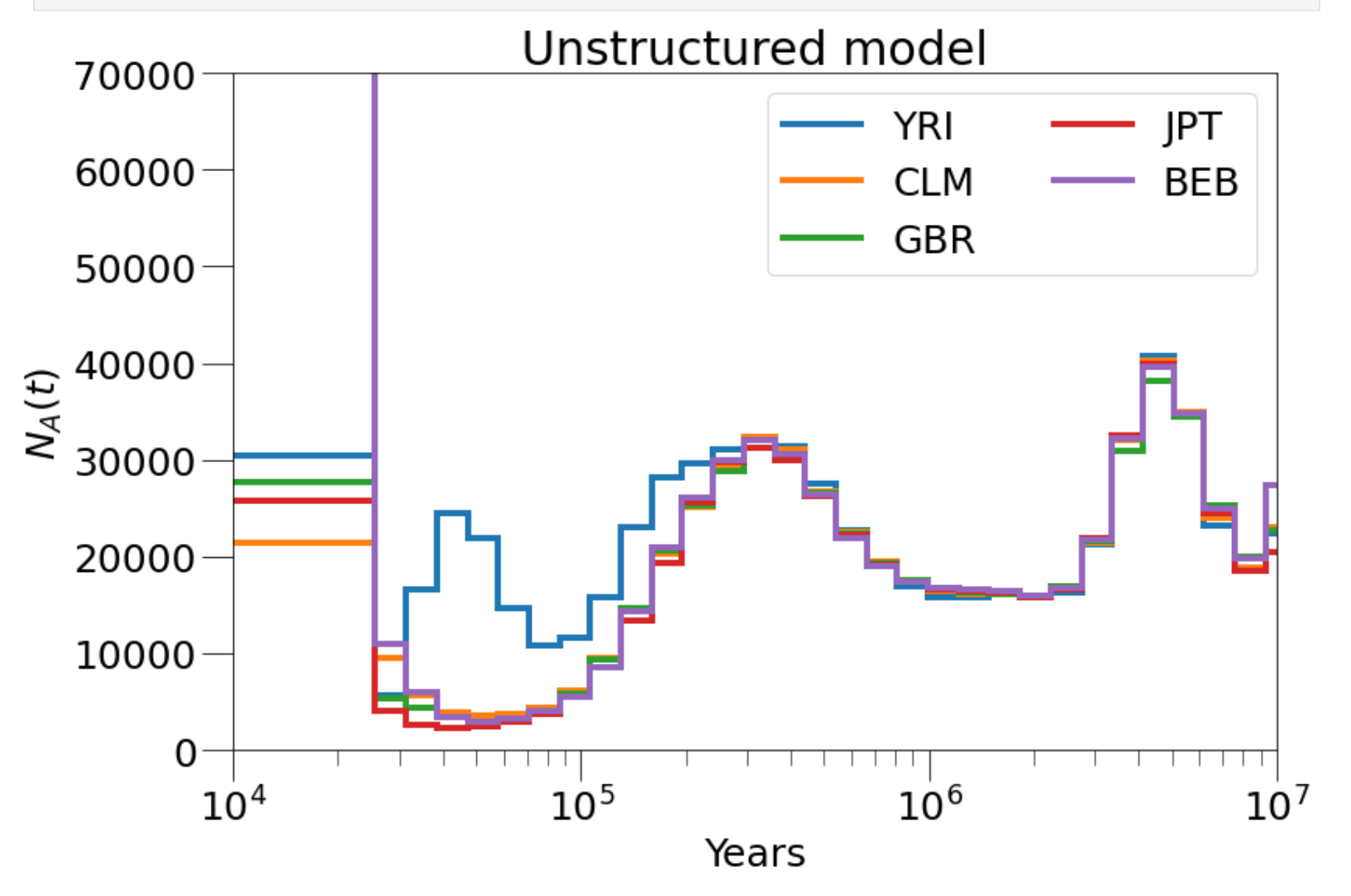
Suppose you have run parameter inference from cobraa, with both a structured model and an unstructured model (i.e. PSMC, which is nested in cobraa). You want to plot inferred model parameters from each, and also see how the structured fit compares to the unstructured fit.

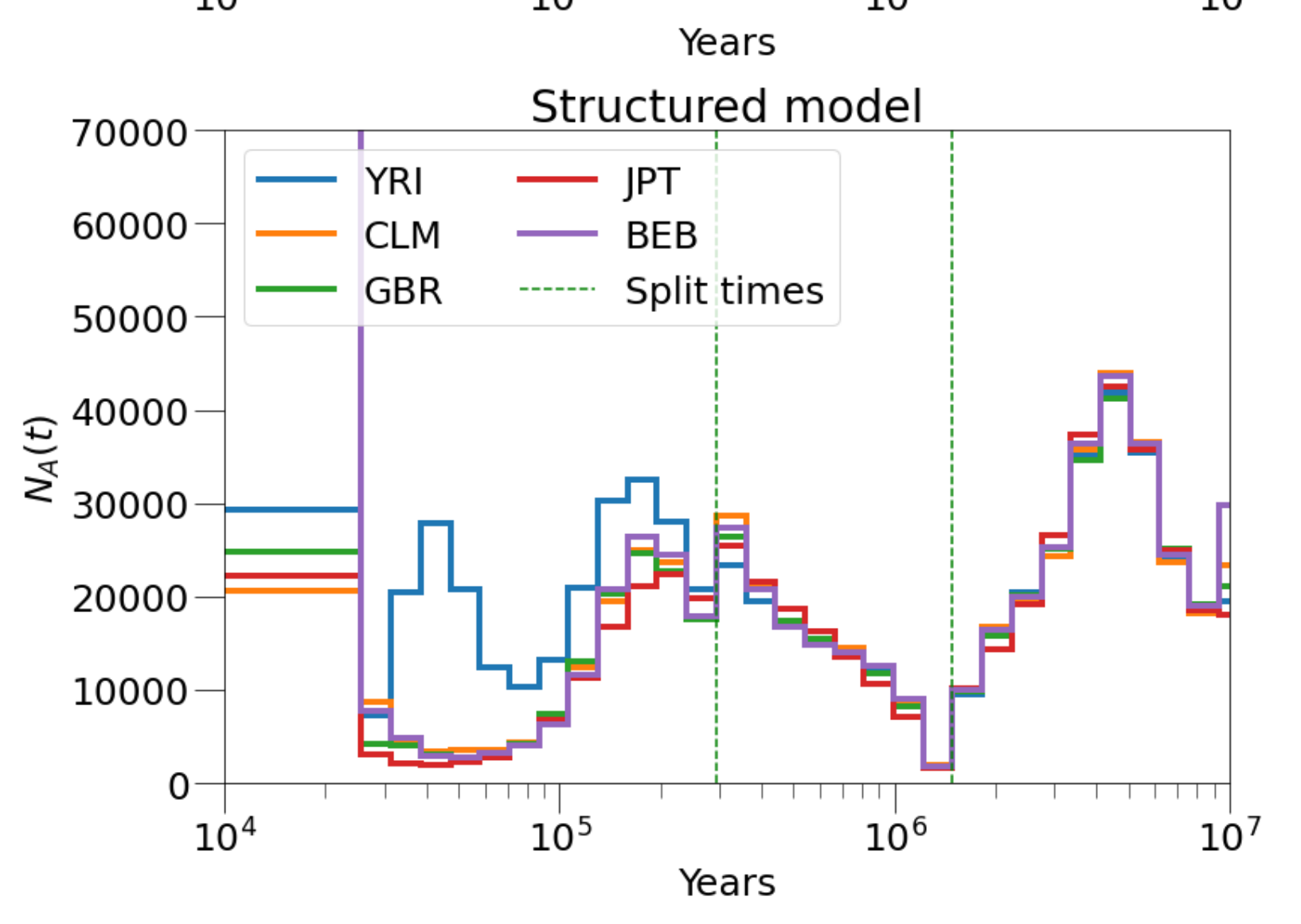
Suppose your structured model fits are in:
/home/trevor/cobraa_tutorial/inferencedata/structure_13_21/YRI_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/structure_13_21/CLM_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/structure_13_21/GBR_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/structure_13_21/JPT_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/structure_13_21/BEB_final_parameters.txt

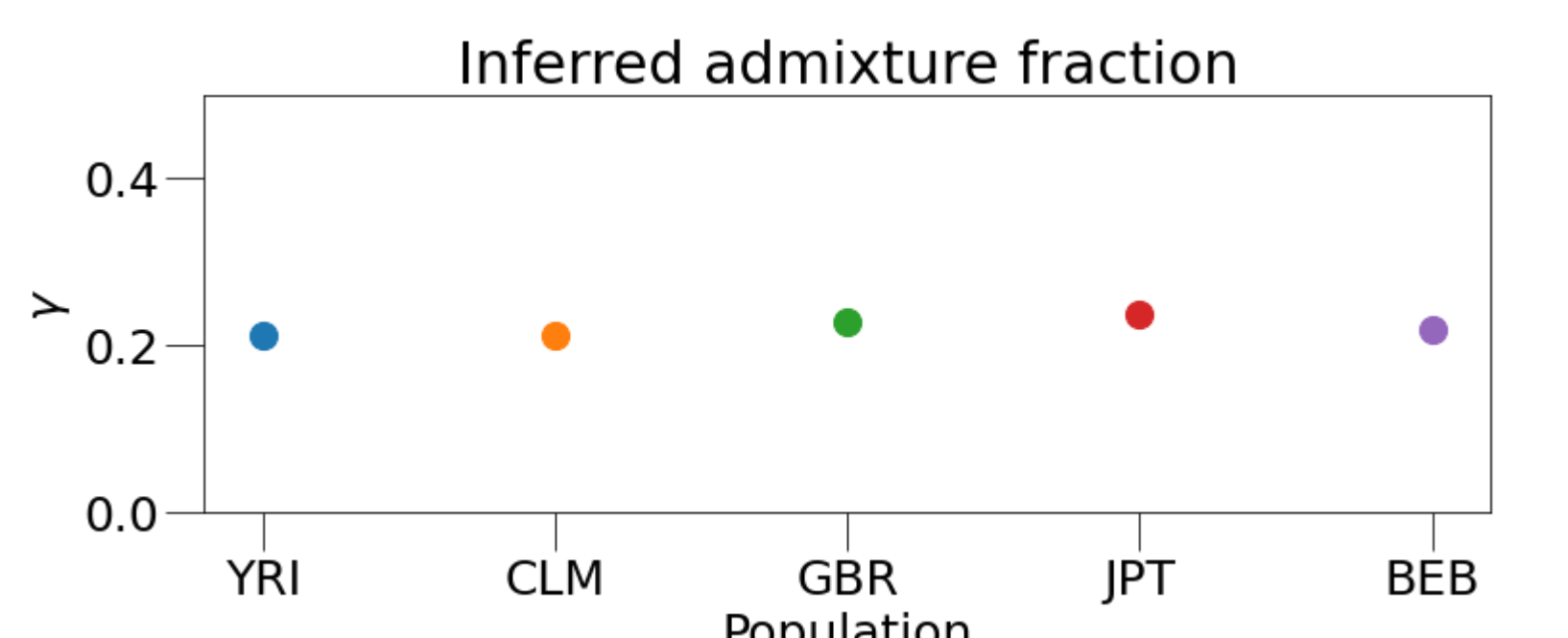
where 13 and 21 refer to the index of T_1 and T_2, respectively, and your unstructured model fits are in:
/home/trevor/cobraa_tutorial/inferencedata/unstructure/YRI_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/unstructure/CLM_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/unstructure/GBR_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/unstructure/JPT_final_parameters.txt
/home/trevor/cobraa_tutorial/inferencedata/unstructure/BEB_final_parameters.txt

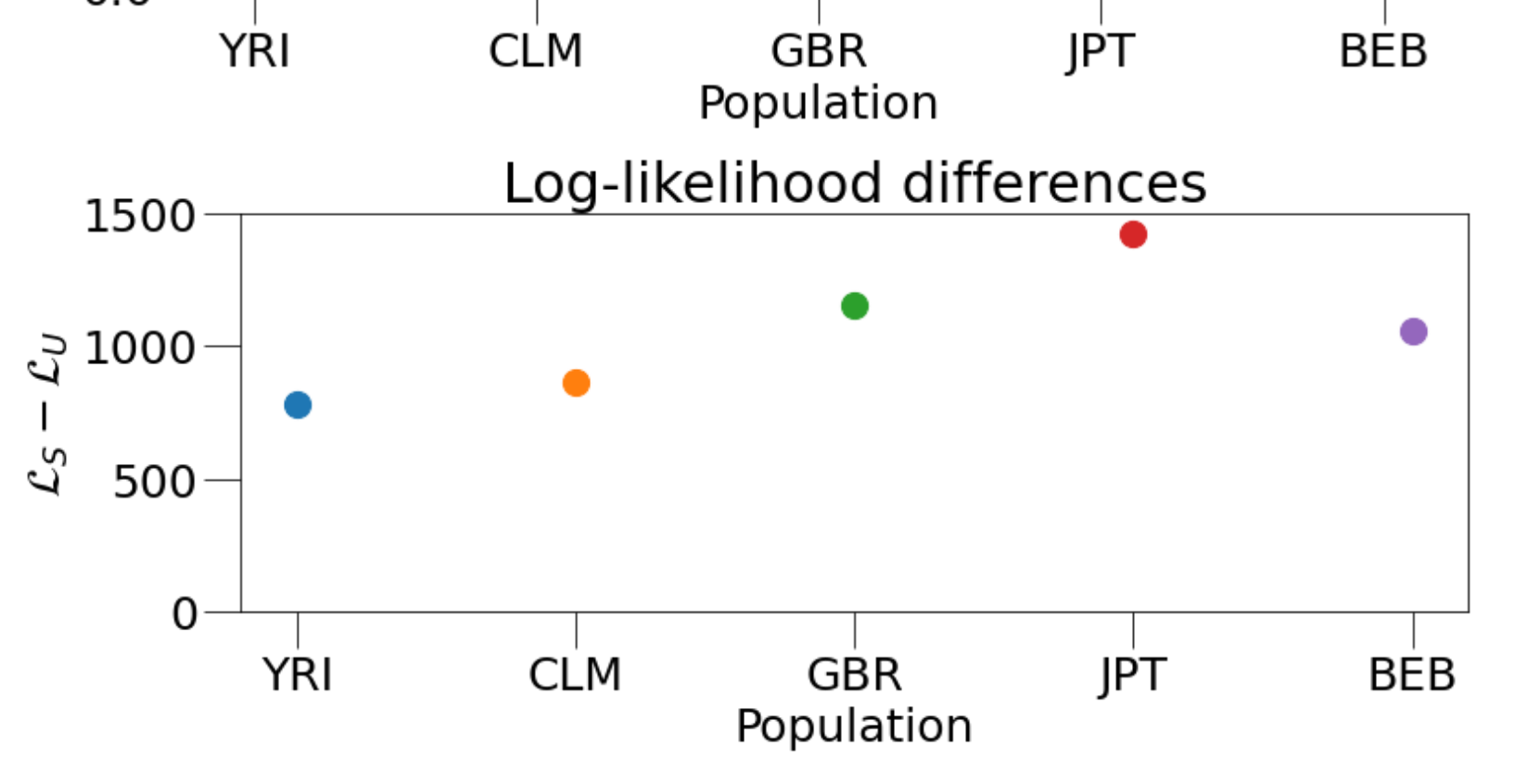
(see github.com/trevorcousins/cobraa/reproducibility for how I generated these).

```
In [37]:
mu = 1.25e-08 #  mutation rate per base pair per generation
gen = 29 # generation time

# unstructured plot (as in PSMC)
for population in ['YRI','CLM','GBR','JPT','BEB']:
    final_params_file = f'/home/tc557/cobraa_tutorial/inferencedata/unstructure/{population}_final_
    final_params = np.loadtxt(final_params_file)
    time_array = list(final_params[:,1])
    time_array.insert(0,0)
    time_array = np.array(time_array)
    plt.stairs(edges=(time_array/mu)*gen,values=(1/final_params[:,2])/mu,label=population,linewidth

plt.xlim(1e+04,1e+07)
plt.ylim(0,7e+04)
plt.xscale('log')
plt.ylabel('$N_A(t)$')
plt.xlabel('Years')
plt.tick_params(which='major',length=20)
plt.tick_params(which='minor',length=10)
plt.legend(ncol=2)
plt.title('Unstructured model')
plt.show()

# structured plot
ts=13
te=21
for population in ['YRI','CLM','GBR','JPT','BEB']:
    final_params_file = f'/home/tc557/cobraa_tutorial/inferencedata/structure_13_21/{population}_fi
    final_params = np.loadtxt(final_params_file)
    time_array = list(final_params[:,1])
    time_array.insert(0,0)
    time_array = np.array(time_array)
    plt.stairs(edges=(time_array/mu)*gen,values=(1/final_params[:,2])/mu,label=population,linewidth
#     plt.stairs(edges=(time_array[ts:te+1]/mu)*gen,values=(1/final_params[ts:te,3])/mu,label='',li
plt.axvline((time_array[ts]/mu)*gen,color="green",linestyle="dashed")
plt.axvline((time_array[te]/mu)*gen,label="Split times",color="green",linestyle="dashed")

plt.xlim(1e+04,1e+07)
plt.ylim(0,7e+04)
plt.xscale('log')
plt.ylabel('$N_A(t)$')
plt.xlabel('Years')
plt.title('Structured model')
plt.tick_params(which='major',length=20)
plt.tick_params(which='minor',length=10)
plt.legend(ncol=2)
plt.show()


# Inferred admixture fraction
plt.rcParams['figure.figsize'] = [12, 4]
plt.rcParams.update({'font.size': 25})
zcount=1
for population in ['YRI','CLM','GBR','JPT','BEB']:
    final_params_file_struct = f'/home/tc557/cobraa_tutorial/inferencedata/structure_13_21/{populat
    gamma = get_gamma_from_file(final_params_file_struct)
    plt.scatter(zcount,gamma,label='',s=200)
    zcount+=1

plt.ylabel('$\gamma$')
plt.xlabel('Population')
plt.title('Inferred admixture fraction')
plt.tick_params(which='major',length=20)
plt.tick_params(which='minor',length=10)
plt.xticks([1,2,3,4,5],['YRI','CLM','GBR','JPT','BEB'])
plt.ylim(0,0.5)

plt.show()

# Differences in likelihood plot
zcount=1
for population in ['YRI','CLM','GBR','JPT','BEB']:
    final_params_file_struct = f'/home/tc557/cobraa_tutorial/inferencedata/structure_13_21/{populat
    final_params_file_pan = f'/home/tc557/cobraa_tutorial/inferencedata/unstructure/{population}_fi
    LL_struct = get_LL_from_file(final_params_file_struct)
    LL_pan = get_LL_from_file(final_params_file_pan)
    LL_diff = LL_struct - LL_pan
    plt.scatter(zcount,LL_diff,label='',s=200)
    zcount+=1

plt.ylabel('$\mathcal{L}_S - \mathcal{L}_U$')
plt.xlabel('Population')
plt.title('Log-likelihood differences')
plt.tick_params(which='major',length=20)
plt.tick_params(which='minor',length=10)
plt.xticks([1,2,3,4,5],['YRI','CLM','GBR','JPT','BEB'])
plt.ylim(0,1500)
plt.show()
```









If you did a search over lots of pairings of T_1 and T_2 (recommended!), then you can visualise how well they fit the data vs panmixia.

```
In [95]:
"""
Searched T_1 and T_2 over the following ranges:
ts_te_pairs = [(None,None)] # (None,None) refers to the panmictic search, as in PSMC
for ts in range(6,14):
    for te in range(17,23):
        ts_te_pairs.append((ts,te))
"""
plt.rcParams['figure.figsize'] = [12, 3]
plt.rcParams.update({'font.size': 25})

fig, axes = plt.subplots(1,5)
zcol=0
for population in ['GBR','JPT','YRI','BEB','CLM']:
    final_params_file_pan = f'/home/tc557/cobraa_tutorial/inferencedata/structure_None_None/{popula
    pan_LL = get_LL_from_file(final_params_file_pan)
    matrix_LL_diffs = np.zeros(shape=(8,6))
    for ts in range(6,14):
        for te in range(17,23):
            final_params_file_struct = f'/home/tc557/cobraa_tutorial/inferencedata/structure_{ts}_{
            struct_LL = get_LL_from_file(final_params_file_struct)
            diff_LL = struct_LL - pan_LL
            matrix_LL_diffs[ts-6,te-17] = diff_LL
    cmaps = OrderedDict()
    cmaps['Diverging'] = [
        'PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu',
        'RdYlBu', 'RdYlGn', 'Spectral', 'coolwarm', 'bwr', 'seismic']
    matrix_LL_diffs = matrix_LL_diffs**3 # rescale to emphasize differences
    sns.heatmap(matrix_LL_diffs,cmap=cmaps['Diverging'][10],ax=axes[zcol],center=0,yticklabels=Fals
    if zcol==0:
        axes[zcol].set_ylabel('Rejoin time ($T_1$)')
    elif zcol==2:
            axes[zcol].set_xlabel('Split time ($T_2$)')
    axes[zcol].set_title(population)
    zcol+=1
```

Analysing the fit of a structured vs unstructured model must be handled with care. The structured model will fit the data just as well or better than the unstructured model, because the unstructured model is nested within the structured model (though we did see in some simulations that the structured model got stuck in local maxima).

Inference of coalescence times and admixture across the genome

We can use cobraa-path to infer regions of the genome that derive from population B. cobraa-path is a HMM where the hidden states are the discretised coalescence time and ancestral lineage path. If t<T_1 then c must be AA. If t>T_1 and t<T_2 then c=AA or c=BB. If t>T_2 then c=AA or c=BB or c=AB. There are thus various ways we can obtain the probability of ancestral lineage path:

1. Marginal probability of ancestral lineage path, $P(c|X)$
2. Probability of ancestral lineage path conditional on the coalescence time being bigger than T_1, $P(c|t>T_1,X)$
3. Probability of ancestral lineage path conditional on the coalescence time being bigger than T_1 and less than T_2, $P(c|T_1<t<T_2,X)$
4. Probability of ancestral lineage path conditional on the coalescence time being bigger than T_2, $P(c|t>T_2,X)$

```
In [97]:
def get_all_A_indices(D_flat,ts,te):
    pre_struct = [i for i in range(0,ts)]
    in_struct = [ts + (j)*2 for j in range(0,te-ts)]
    post_struct = [ts + (te-ts)*2 + (j)*3 for j in range(0,D-te)]
    all_A_indices = pre_struct + in_struct + post_struct
    return all_A_indices

def get_all_B_indices(D_flat,ts,te):
    in_struct = [ts + ((j)*2+1) for j in range(0,te-ts)]
    post_struct = [ts + (te-ts)*2+ (j)*3+1 for j in range(0,D-te)]
    all_B_indices = in_struct + post_struct
    return all_B_indices


def get_AB_poststruct_indices(D_flat,ts,te):
    AB_indices = [ts + (te-ts)*2 + (j)*3+2 for j in range(0,D-te)]
    return AB_indices

D = 32 # number of discrete time intervals in the HMM
T_1 = 13 # composite ML estimate of admixture time
T_2 = 21 # composite ML estimate of divergence time
T_2_flat = T_1+(T_2-T_1)*2 # index of T_2 in flatten HMM transition matrix
D_flat = T_1 + (T_2-T_1)*2 + (D-T_2)*3 # number of indices in posterior decoding; there are 32 time
all_A = get_all_A_indices(D_flat,T_1,T_2) # indices of c=AA
all_B = get_all_B_indices(D_flat,T_1,T_2) # indices of c=BB ( implicitly, t>T_1 for c=BB to have no
all_AB = get_AB_poststruct_indices(D_flat,T_1,T_2) # indices of c=AB
all_A_given_t_bigger_ts = [i for i in all_A if i>=T_1] # indices of c=AA and t>T_1
all_A_given_t_bigger_te = [i for i in all_A if i>=T_2_flat] # indices of c=AA and t>T_2
all_B_given_t_bigger_te = [i for i in all_B if i>=T_2_flat] # indices of c=B and t>T_2
all_AB_given_t_bigger_te = [i for i in all_AB if i>=T_2_flat] # indices of c=AB and t>T_2
all_A_given_t_bigger_ts_smaller_te = [i for i in all_A if i>=T_1 and i<T_2_flat] # indices of c=AA
all_B_given_t_bigger_ts_smaller_te = [i for i in all_B if i>=T_1 and i<T_2_flat] # indices of c=BB
t_in_structured_period = [i for i in range(T_1,T_2_flat)] # index of t for structured period
step_size=1000 # length between base pairs of posterior decoding
```

```
In [100…
chrom=20
popsam='ESN_HG03515'
decode_file = f'/home/tc557/rds/hpc-work/copy_to_Zenodo_240315/decoding/D_32/b_100/spread1_0.075/sp
decode = np.loadtxt(decode_file) # load file
decode_position = decode[0,:] # get the genomic position
posterior = decode[1:,:] # get the full posterior, this must sum to 1

prob_AA_marginal = posterior[all_A,:].sum(axis=0) # P(c=AA|X)
prob_BB_marginal = posterior[all_B,:].sum(axis=0) # P(c=BB|X)
prob_AB_marginal = posterior[all_AB,:].sum(axis=0) # # P(c=AB|X)
prob_AA_givent_bigger_T1 = posterior[all_A_given_t_bigger_ts,:].sum(axis=0)/posterior[np.arange(T_1
prob_BB_givent_bigger_T1 = posterior[all_B,:].sum(axis=0)/posterior[np.arange(T_1,D_flat,1),:].sum(
prob_AB_givent_bigger_T1 = posterior[all_AB,:].sum(axis=0)/posterior[np.arange(T_1,D_flat,1),:].sum
prob_AA_givent_bigger_T2 = posterior[all_A_given_t_bigger_te,:].sum(axis=0)/posterior[np.arange(T_2
prob_BB_givent_bigger_T2 = posterior[all_B_given_t_bigger_te,:].sum(axis=0)/posterior[np.arange(T_2
prob_AB_givent_bigger_T2 = posterior[all_AB_given_t_bigger_te,:].sum(axis=0)/posterior[np.arange(T_
prob_AA_givent_bigger_T1_smaller_T2 = posterior[all_A_given_t_bigger_ts_smaller_te,:].sum(axis=0)/p
prob_BB_givent_bigger_T1_smaller_T2 = posterior[all_B_given_t_bigger_ts_smaller_te,:].sum(axis=0)/p
```

```
In [146…
# marginal probabilities of paths

start = 10000
end = 20000
num_xticks = 5
xtickslocs = np.linspace(start,end,num_xticks) - start
xtickslabs = [str(i/1000000)+'Mb' for i in np.linspace(start,end,num_xticks)*step_size]

plt.rcParams['figure.figsize'] = [24, 12]
plt.rcParams.update({'font.size': 25})

np.linspace(start,end,num_xticks)*step_size

# full posterior (coalescence times and lineage paths)
ax = sns.heatmap(posterior[:,start:end],cbar_kws={'label': 'Probability'})
# ax.set_yticks(ticks=ydom,labels=yticks_labs,rotation='horizontal')
ax.set_xticks(ticks=xtickslocs,labels=xtickslabs,rotation='horizontal')
ax.set_xlabel('Chromosome')
ax.set_ylabel('Coalescence times and ancestral lineage path')
ax.invert_yaxis()
ax.set_title('Full posterior')

plt.show()

plt.rcParams['figure.figsize'] = [24, 4]
plt.rcParams.update({'font.size': 25})


# No conditioning
probs = np.array([prob_AA_marginal,prob_BB_marginal,prob_AB_marginal])
ax = sns.heatmap(probs[:,start:end],cbar_kws={'label': 'Probability'})
# ax.set_yticks(ticks=ydom,labels=yticks_labs,rotation='horizontal')
ax.set_xticks(ticks=xtickslocs,labels=xtickslabs,rotation='horizontal')
ax.set_xlabel('Chromosome')
ax.set_ylabel('Ancestral\nlineage path')
ax.invert_yaxis()
ax.set_title('Marginal path probabilities')
plt.show()


# conditioning on t > T_1
probs_condt1 = np.array([prob_AA_givent_bigger_T1,prob_BB_givent_bigger_T1,prob_AB_givent_bigger_T1
ax = sns.heatmap(probs_condt1[:,start:end],cbar_kws={'label': 'Probability'})
# ax.set_yticks(ticks=ydom,labels=yticks_labs,rotation='horizontal')
ax.set_xticks(ticks=xtickslocs,labels=xtickslabs,rotation='horizontal')
plt.yticks([0.5,1.5,2.5],['AA','BB','AB'])
ax.set_xlabel('Chromosome')
ax.set_ylabel('Ancestral\nlineage path')
ax.invert_yaxis()
ax.set_title('Marginal path probabilities (conditional on $t>T_1$)')
plt.show()

# conditioning on t > T_2
#     probs_condt2 = np.array([prob_AA_givent_bigger_T2,prob_BB_givent_bigger_T2,prob_AB_givent_big
#     ax = sns.heatmap(probs_condt2[:,start:end],cbar_kws={'label': 'Probability'})
#     # ax.set_yticks(ticks=ydom,labels=yticks_labs,rotation='horizontal')
#     ax.set_xticks(ticks=xtickslocs,labels=xtickslabs,rotation='horizontal')
#     plt.yticks([0.5,1.5,2.5],['AA','BB','AB'])
#     ax.set_xlabel('Chromosome')
#     ax.set_ylabel('Ancestral\nlineage path')
#     ax.invert_yaxis()
#     ax.set_title('Marginal path probabilities (conditional on $t>T_2$)')
#     plt.show()

# conditioning on T_1 < t < T_2
#     probs_condt1t2 = np.array([prob_AA_givent_bigger_T1_smaller_T2,prob_BB_givent_bigger_T1_small
#     ax = sns.heatmap(probs_condt1t2[:,start:end],cbar_kws={'label': 'Probability'})
#     # ax.set_yticks(ticks=ydom,labels=yticks_labs,rotation='horizontal')
#     ax.set_xticks(ticks=xtickslocs,labels=xtickslabs,rotation='horizontal')
#     plt.yticks([0.5,1.5],['AA','BB'])
#     ax.set_xlabel('Chromosome')
#     ax.set_ylabel('Ancestral\nlineage path')
#     ax.invert_yaxis()
#     ax.set_title('Marginal path probabilities (conditional on $T_1<t<T_2$)')
#     plt.show()
```