

Supplementary Material (B) - Testing the Circumplex Structure of the Soundscape Survey

Andrew Mitchell^{a,*}, Francesco Aletta^a

^aUniversity College London, Institute for Environmental Design and Engineering, Central House, 14 Upper Woburn Place, London, WC1H 0NN

1. Step 2: SEM Fit Score

```
# Import the required packages
import pandas as pd
import seaborn as sns
from pathlib import Path
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy as np
from datetime import datetime
import circumplex
import json
from scipy.spatial import procrustes
from tensorly.metrics.factors import congruence_coefficient
```

```
today = datetime.today().strftime('%Y-%m-%d')
```

```
# Define the scales and angles to be used
scales = ["PAQ1", "PAQ2", "PAQ3", "PAQ4", "PAQ5", "PAQ6", "PAQ7", "PAQ8"]
eq_angles = [0, 45, 90, 135, 180, 225, 270, 315]
```

```
# Define the data and output folders
data_folder = Path("../data/")
output_folder = Path(f"../outputs/{today}")
```

```
# Load data
satp = pd.read_excel(data_folder / "SATP Dataset v1.4.xlsx")
```

```
lvls = pd.read_excel(data_folder / "LLAN.xlsx")
# Clean up the lvls data
lvls = lvls.groupby("Mark/Group Name").max().drop("Channel Name", axis=1)
lvls.rename(columns={"L/dB(SPL)": "max_Leq", "L(A)/dB(SPL)": "max_LAeq", "N/soneGF": "max_N", "L90(A)"}).
```

*Corresponding author

Email addresses: andrew.mitchell.18@ucl.ac.uk (Andrew Mitchell), f.aletta@ucl.ac.uk (Francesco Aletta)

```

# Add the levels to the satp data
satp = satp.merge(lvls, left_on="Recording", right_on = "Mark/Group Name", right_index=True)

# Load the results from the latest SEM analysis
sem_res = pd.read_csv(output_folder / "sem-fit-ipsatized.csv")
sem_res.drop("Unnamed: 0", axis=1, inplace=True)

# In some cases, the SEM flips the angles (i.e. vibrant is at 315 degrees instead of 45).
# This function checks for this and corrects it, to ensure all the scales are in the
# correct order, but without changing the relationship between the angles, as identified by the SEM.)

# First, get the angles from the SEM results
ang_df = sem_res[sem_res['Model Type'] == 'Equal comm.'][["Language"] + scales]
ang_df.set_index("Language", inplace=True)

def check_inverse_angles(language_angles):
    """Check if the angles are inverse"""
    if language_angles[1] > language_angles[2] or language_angles[2] > language_angles[3]:
        return True
    else:
        return False

# Then, check if the angles are inverse, and if so, correct them
for lang in ang_df.index:
    if check_inverse_angles(ang_df.loc[lang].values):
        ang_df.loc[lang][1:] = 360 - ang_df.loc[lang][1:]

ang_dict = ang_df.T.to_dict(orient="list")
ang_dict

{'eng': [0.0, 46.0, 94.0, 138.0, 177.0, 231.0, 275.0, 340.0],
 'arb': [0.0, 36.0, 45.0, 135.0, 167.0, 201.0, 242.0, 308.0],
 'cmn': [0.0, 18.0, 38.0, 154.0, 171.0, 196.0, 217.0, 318.0],
 'deu': [0.0, 64.0, 97.0, 132.0, 182.0, 254.0, 282.0, 336.0],
 'ell': [0.0, 72.0, 86.0, 133.0, 161.0, 233.0, 267.0, 328.0],
 'fra': [0.0, 69.0, 102.0, 129.0, 173.0, 246.0, 275.0, 325.0],
 'hrv': [0.0, 84.0, 93.0, 160.0, 173.0, 243.0, 273.0, 354.0],
 'ind': [0.0, 53.0, 104.0, 123.0, 139.0, 202.0, 284.0, 308.0],
 'ita': [0.0, 57.0, 104.0, 142.0, 170.0, 274.0, 285.0, 336.0],
 'jpn': [0.0, 46.0, 101.0, 138.0, 159.0, 271.0, 288.0, 339.0],
 'kor': [0.0, 56.0, 90.0, 124.0, 151.0, 251.0, 275.0, 288.0],
 'nld': [0.0, 43.0, 111.0, 125.0, 174.0, 257.0, 307.0, 341.0],
 'por': [0.0, 81.0, 121.0, 140.0, 171.0, 252.0, 275.0, 334.0],
 'spa': [0.0, 41.0, 103.0, 147.0, 174.0, 238.0, 279.0, 332.0],
 'swe': [0.0, 66.0, 87.0, 146.0, 175.0, 249.0, 275.0, 335.0],
 'tur': [0.0, 55.0, 97.0, 106.0, 157.0, 254.0, 289.0, 313.0]}

ang_dict['eng']

[0.0, 46.0, 94.0, 138.0, 177.0, 231.0, 275.0, 340.0]

```

1.0.1. Calculate the SEM fit score

The SEM fit score is calculated by counting the number of fit indices that pass the pre-defined threshold. The thresholds are defined in the first part of the code. The thresholds are based on the thresholds used by Rogoza et al. (2021).

The first part of the code defines a dictionary named **thresholds** which contains the thresholds for different fit criteria used in SEM. These criteria include p-value (p), Comparative Fit Index (CFI), Goodness of Fit Index (GFI), Adjusted Goodness of Fit Index (AGFI), Standardized Root Mean Square Residual (SRMR), and others.

The **incl_in_score** list is used to select which criteria will be included in the final score calculation. The **pass_thresh** and **tent_thresh** variables define the thresholds for passing and tentative passing scores, respectively.

The next part of the code calculates whether each SEM result passes the defined thresholds for each criterion. This is done by comparing the SEM result for each criterion to its respective threshold. The results of these comparisons are stored as boolean values in new columns in the **sem_res** DataFrame.

The final score for each SEM result is then calculated by summing the number of criteria each result passes. This score is stored as an integer in a new column in the **sem_res** DataFrame. The passing column categorizes each SEM result as 'Fail', 'Tentative', or 'Pass' based on its final score.

Finally, the results are saved to an Excel file and a subset of the results is displayed. The subset includes only the results for the "Equal comm." model type and is sorted by score in descending order.

```
# Define the thresholds for the SEM fit criteria
thresholds = {
    "CFI": 0.92, # Moshona et al 2023, Hu & Bentler 1999
    # "CFI": 0.9, # Rogoza 2021
    # "CFI": 0.95, # Good fit from Tarlao et al 2021
    "GFI": 0.9, # Rogoza 2021
    # "AGFI": 0.85, # Rogoza 2021
    "SRMR": 0.08, # Moshona et al 2023, Tarlao et al 2021, West (2023), Hu & Bentler (1999)
    # "MCSC": -0.7,
    # Removed RMSEA at the suggestion of West, S.G., Wu, W., McNeish, D., & Savord, A. (2023). Model F
}

# Choose which criteria to include in the final score
# incl_in_score = ['CFI', 'GFI', 'AGFI', 'RMSEA'] # Rogoza
incl_in_score = ['CFI', 'GFI', 'SRMR']

# Define the thresholds for the final score
pass_thresh = 3

# Calculate the final score
sem_res['CFI_pass'] = sem_res['CFI'] >= thresholds['CFI']
sem_res['GFI_pass'] = sem_res['GFI'] >= thresholds['GFI']
sem_res['SRMR_pass'] = sem_res['SRMR'] <= thresholds['SRMR']

sem_res['Score'] = sem_res[[x + '_pass' for x in incl_in_score]].sum(axis=1)
sem_res['Score'] = sem_res['Score'].astype(int)
sem_res['passing'] = pd.cut(sem_res['Score'], bins=[0, pass_thresh, 7], labels=['Fail', 'Pass'], right=True)
# Save the results
sem_res.to_excel(output_folder / f"sem-fit-results.xlsx", index=False)
```

```
sem_res[["Language", "Model Type", "n", "m", "CFI", "GFI", "SRMR", "Score", "passing"]].loc[sem_res["M
```

Table 1: SEM fit results

| | Language | Model Type | n | m | CFI | GFI | SRMR | Score | passing |
|----|----------|-------------|------|---|------|------|------|-------|---------|
| 1 | eng | Equal comm. | 864 | 3 | 0.93 | 0.91 | 0.05 | 3 | Pass |
| 5 | arb | Equal comm. | 809 | 3 | 0.97 | 0.97 | 0.04 | 3 | Pass |
| 9 | cmn | Equal comm. | 1832 | 3 | 0.96 | 0.95 | 0.04 | 3 | Pass |
| 13 | deu | Equal comm. | 810 | 3 | 0.94 | 0.92 | 0.06 | 3 | Pass |
| 17 | ell | Equal comm. | 810 | 3 | 0.93 | 0.93 | 0.08 | 3 | Pass |
| 25 | hrv | Equal comm. | 864 | 3 | 0.95 | 0.93 | 0.06 | 3 | Pass |
| 29 | ind | Equal comm. | 891 | 3 | 0.93 | 0.92 | 0.08 | 3 | Pass |
| 33 | ita | Equal comm. | 810 | 3 | 0.94 | 0.93 | 0.07 | 3 | Pass |
| 41 | kor | Equal comm. | 810 | 3 | 0.95 | 0.94 | 0.08 | 3 | Pass |
| 45 | nld | Equal comm. | 864 | 3 | 0.97 | 0.94 | 0.06 | 3 | Pass |
| 53 | spa | Equal comm. | 1647 | 3 | 0.92 | 0.91 | 0.06 | 3 | Pass |
| 57 | swe | Equal comm. | 945 | 3 | 0.94 | 0.92 | 0.05 | 3 | Pass |
| 61 | tur | Equal comm. | 918 | 3 | 0.93 | 0.92 | 0.08 | 3 | Pass |
| 21 | fra | Equal comm. | 891 | 3 | 0.92 | 0.91 | 0.10 | 2 | Fail |
| 49 | por | Equal comm. | 1890 | 3 | 0.92 | 0.92 | 0.09 | 2 | Fail |
| 37 | jpn | Equal comm. | 917 | 3 | 0.89 | 0.90 | 0.09 | 1 | Fail |

```
passing = sem_res.loc[sem_res["Model Type"] == "Equal comm."].query("passing != 'Fail'")['Language'].v
satp = satp.query("Language in @passing")
```

1.0.2. Steps 3 and 4: Accurately locating circumplex items within each language

For this analysis, we use a custom Python package developed for this paper, called `circumplex` which can be installed from PyPI.

The process to do this is as follows:

1. For both the reference and test circumplex, calculate the mean value of each scale for each recording.
2. Calculate the SSM correlation between each scale of the reference circumplex and the test circumplex, in our case using the corrected angles.
3. Test the congruence between the empirical locations and theoretical expectations using the Procrustes congruence test (Rogoza et al., 2021).

We will be using the full dataset as the reference set and the data from each translation as the test set. This effectively means that we are testing whether each translation is able to locate the circumplex structure of the soundscape survey, consistently across all languages.

1.0.2.1. Congruence.

What Rogoza et al. (2021) (and Orthosim) refer to as Tucker’s Congruence Coefficient is also commonly referred to as the cosine similarity (see the [Tensorly documentation](#)). We therefore use the sklearn implementation of cosine similarity to calculate the congruence between the empirical locations and theoretical expectations. This produces a matrix of cosine similarity values, which we then use to calculate the mean congruence, to match the model congruence from Rogoza et al. (2021).

We can confirm the equivalence of this method by comparing with the results from Rogoza et al. (2021)’s Orthosim analysis:

```
def congruence(data1, data2, metric: ['cosine', 'euclidean']='cosine'):
    from sklearn.metrics.pairwise import cosine_similarity
    from scipy.spatial.distance import cdist

    if metric == 'cosine':
        sim = cosine_similarity(data1, data2)
    elif metric == 'euclidean':
        sim = 1 - cdist(data1, data2, metric=metric)
    else:
        raise ValueError("metric must be 'cosine' or 'euclidean'")
    cong_vals = np.diag(sim) # Get the diagonal values which compare against the appropriate angles
    return np.mean(cong_vals), cong_vals

# Data from Rogoza
data2 = np.array([[ -0.59, -0.8], [-1, -0.05], [-0.5, 0.87]])
data1 = np.array([[ -0.71, -0.71], [-1, 0], [-0.71, 0.71]])

model_congruence, scale_congruence = congruence(data1, data2)

print("Model Congruence == 0.984: ", np.round(model_congruence, 3) == 0.984)
print("Vulnerability Congruence == 0.99: ", np.round(scale_congruence[0], 2) == 0.99)
print("Antagonism Congruence == 1.0: ", np.round(scale_congruence[1], 2) == 1.0)
print("Grandiosity Congruence == 0.97: ", np.round(scale_congruence[2], 2) == 0.97)
```

```
Model Congruence == 0.984: True
Vulnerability Congruence == 0.99: True
Antagonism Congruence == 1.0: True
Grandiosity Congruence == 0.97: True
```

1.0.2.2. Procrustes.

However, it appears that, despite [Rogoza et al. \(2021\)](#) 's description, this method is not actually based on a Procrustes analysis. The equivalent distance metric from a Procrustes method would be the rotational-based Procrustes distance, i.e. the squared Frobenius norm of the difference between the two orthogonal matrices. See [Andreella et al. \(2023\)](#):

Instead, the second distance exploits the orthogonal matrix parameters solution of the Procrustes problem. The rotational-based distance computes the squared Frobenius distance between these estimated orthogonal matrices. As we will see, this metric measures the level of dissimilarity/similarity in orientation between matrices/subjects before functional alignment.

As such, we can also calculate this distance using the [procrustes](#) package:

```
def procrustes_distance(data1, data2, procrustes_type='orthogonal', translate=True, scale=True):
    if procrustes_type == 'orthogonal':
        from procrustes import orthogonal
        pro_res = orthogonal(data1, data2, translate=translate, scale=scale)
    elif procrustes_type == 'rotational':
        from procrustes import rotational
        pro_res = rotational(data1, data2, translate=translate, scale=scale)
    elif procrustes_type == 'generic':
        from procrustes import generic
        pro_res = generic(data1, data2, translate=translate, scale=scale)
    return pro_res.error
```

```

def prepare_congruence_matrices(ssm_table, target_angles = eq_angles):
    """Prepare the congruence matrices for the SSM results"""
    # Get the data from the SSM table
    # data2 = np.array((np.cos(np.deg2rad(ssm_table['displacement'])), np.sin(np.deg2rad(ssm_table['displacement']))))
    data2 = ssm_table[['xval', 'yval']].values
    # Get the data from the target angles
    data1 = np.ones_like(data2)
    data1[:, 0] = np.cos(np.deg2rad(target_angles))
    data1[:, 1] = np.sin(np.deg2rad(target_angles))

    return data1, data2

```

Based on the proof given in ?, given that the input matrices are scaled (i.e. `rotational(scale=True)`), then the Procrustes distance is a true distance measure which obeys $0 < p(X, Y) < 1$ (Bakhtiar and Siswadi, 2015, 322). Therefore we can convert the Procrustes distance to a similarity measure by subtracting it from 1.

```

overall_means = satp.groupby(["Recording"])[scales].mean().reset_index(drop=True)

lang_rec_means = satp.groupby(["Language", "Recording"])[scales].mean()
lang_rec_means.reset_index(inplace=True)

def test_language_locations(test_lang_means, test_lang, test_angles, target_angles=eq_angles, scales=scales):
    """Test the congruence of the language locations with the target angles"""
    # test_lang_means = lang_rec_means.query("Language == @test_lang")[scales].reset_index(drop=True)
    test_res = []
    for scale in scales:
        corrs = test_lang_means.corrwith(overall_means[scale])
        ssm_res = circumplex.SSMParams(
            corrs,
            scales,
            test_angles,
            scale,
        )
        test_res.append(ssm_res)

    test_ssm = circumplex.SSMResults(test_res)

    data1, data2 = prepare_congruence_matrices(test_ssm.table, target_angles=target_angles)
    test_model_congruence, test_scale_congruences = congruence(data1, data2, metric='cosine')
    pro_sim = 1 - procrustes_distance(data1, data2, procrustes_type='rotational')
    return test_ssm, test_model_congruence, test_scale_congruences, pro_sim

locating_corr_angles = {}
locating_eq_angles = {}
for test_lang in lang_rec_means.Language.unique():
    test_lang_means = lang_rec_means.query("Language == @test_lang")[scales].reset_index(drop=True)

    locating_eq_angles[test_lang] = test_language_locations(test_lang_means, test_lang, test_angles=eq_angles)
    locating_corr_angles[test_lang] = test_language_locations(test_lang_means, test_lang, test_angles=scales)

```

1.0.3. Step 3 Results

```
fit_results = pd.DataFrame.from_dict({key: locating_corr_angles[key][0].table.r2 for key in locating_corr_angles.keys()})
fit_results.round(3)
```

Table 2: Fit results for locating the general circumplex within each language

| | PAQ1 | PAQ2 | PAQ3 | PAQ4 | PAQ5 | PAQ6 | PAQ7 | PAQ8 |
|-----|-------|-------|-------|-------|-------|-------|-------|-------|
| arb | 0.998 | 0.999 | 0.995 | 0.995 | 0.998 | 0.999 | 0.997 | 0.995 |
| cmn | 0.997 | 0.996 | 0.988 | 0.993 | 0.999 | 0.999 | 0.989 | 0.992 |
| deu | 0.998 | 0.999 | 0.999 | 0.998 | 0.998 | 0.998 | 0.999 | 0.998 |
| ell | 0.998 | 0.995 | 0.998 | 0.998 | 0.997 | 0.998 | 0.996 | 0.997 |
| eng | 0.997 | 0.998 | 0.999 | 0.996 | 0.997 | 0.999 | 0.998 | 0.994 |
| hrv | 0.997 | 0.997 | 0.994 | 0.995 | 0.997 | 0.998 | 0.995 | 0.991 |
| ind | 0.996 | 0.998 | 0.998 | 0.998 | 0.996 | 0.993 | 0.999 | 0.996 |
| ita | 0.892 | 0.996 | 0.998 | 0.986 | 0.980 | 0.991 | 0.996 | 0.977 |
| kor | 0.997 | 0.996 | 0.996 | 0.998 | 0.998 | 0.991 | 0.997 | 0.998 |
| nld | 0.993 | 0.999 | 0.999 | 0.998 | 0.998 | 0.996 | 0.999 | 0.999 |
| spa | 0.998 | 0.996 | 0.998 | 0.998 | 0.998 | 0.996 | 0.999 | 0.999 |
| swe | 0.994 | 0.998 | 0.997 | 0.997 | 0.996 | 0.997 | 0.997 | 0.996 |
| tur | 0.997 | 0.997 | 0.998 | 0.999 | 0.996 | 0.993 | 0.999 | 0.997 |

1.0.4. Step 4 Results

```
congruence_df = pd.DataFrame(
    {
        "Language": list(locating_eq_angles.keys()),
        "Eq Ang Model": [x[1] for x in locating_eq_angles.values()],
        "Corr Ang Model": [x[1] for x in locating_corr_angles.values()],
        "Eq Ang Procrustes": [x[3] for x in locating_eq_angles.values()],
        "Corr Ang Procrustes": [x[3] for x in locating_corr_angles.values()],
    }
)
congruence_df.round(3)
```

Table 3: Correspondence between the general circumplex and the language-specific circumplex

| | Language | Eq Ang Model | Corr Ang Model | Eq Ang Procrustes | Corr Ang Procrustes |
|----|----------|--------------|----------------|-------------------|---------------------|
| 0 | arb | 0.992 | 0.943 | 0.982 | 0.982 |
| 1 | cmn | 0.932 | 0.993 | 0.885 | 0.991 |
| 2 | deu | 0.985 | 0.957 | 0.973 | 0.983 |
| 3 | ell | 0.985 | 0.989 | 0.970 | 0.979 |
| 4 | eng | 0.988 | 0.977 | 0.983 | 0.983 |
| 5 | hrv | 0.990 | 0.969 | 0.982 | 0.985 |
| 6 | ind | 0.965 | 0.951 | 0.935 | 0.982 |
| 7 | ita | 0.984 | 0.952 | 0.974 | 0.975 |
| 8 | kor | 0.958 | 0.977 | 0.921 | 0.975 |
| 9 | nld | 0.978 | 0.972 | 0.939 | 0.978 |
| 10 | spa | 0.986 | 0.981 | 0.967 | 0.978 |
| 11 | swe | 0.986 | 0.973 | 0.972 | 0.976 |
| 12 | tur | 0.957 | 0.986 | 0.921 | 0.980 |

1.0.5. Profile Plots

```
for res in locating_corr_angles['cmn'][0].results:
    res.profile_plot()
plt.show()
```

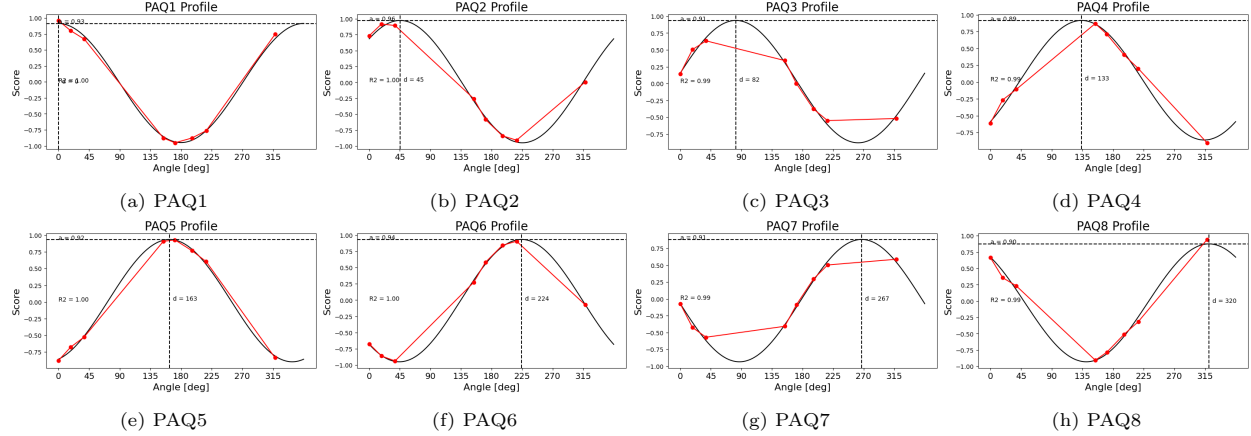


Figure 1: Profile plots for the Mandarin (cmn) translation

```
from matplotlib import colormaps

fig, ax = plt.subplots(subplot_kw=dict(projection='polar'))
colors = colormaps.get_cmap('tab10').colors
colors = iter(colors)
for res in locating_eq_angles['cmn'][0].results:
    ax.plot(
        np.deg2rad(res.displacement),
        res.amplitude,
        color=next(colors),
        marker='o',
        markersize=10,
        label=res.label,
    )

# locating_eq_angles['cmn'][0].plot(legend=False)
plt.show()
# locating_corr_angles['cmn'][0].plot()

fig, ax = plt.subplots(subplot_kw=dict(projection='polar'))
colors = colormaps.get_cmap('tab10').colors
colors = iter(colors)
for res in locating_corr_angles['cmn'][0].results:
    ax.plot(
        np.deg2rad(res.displacement),
        res.amplitude,
        color=next(colors),
        marker='o',
        markersize=10,
```



```

        label=res.label,
    )
ax.set_thetagrids(locating_corr_angles['cmn'][0].results[0].angles)
ax.legend(loc='upper right', bbox_to_anchor=(1.4, 1.0))
plt.show()

```

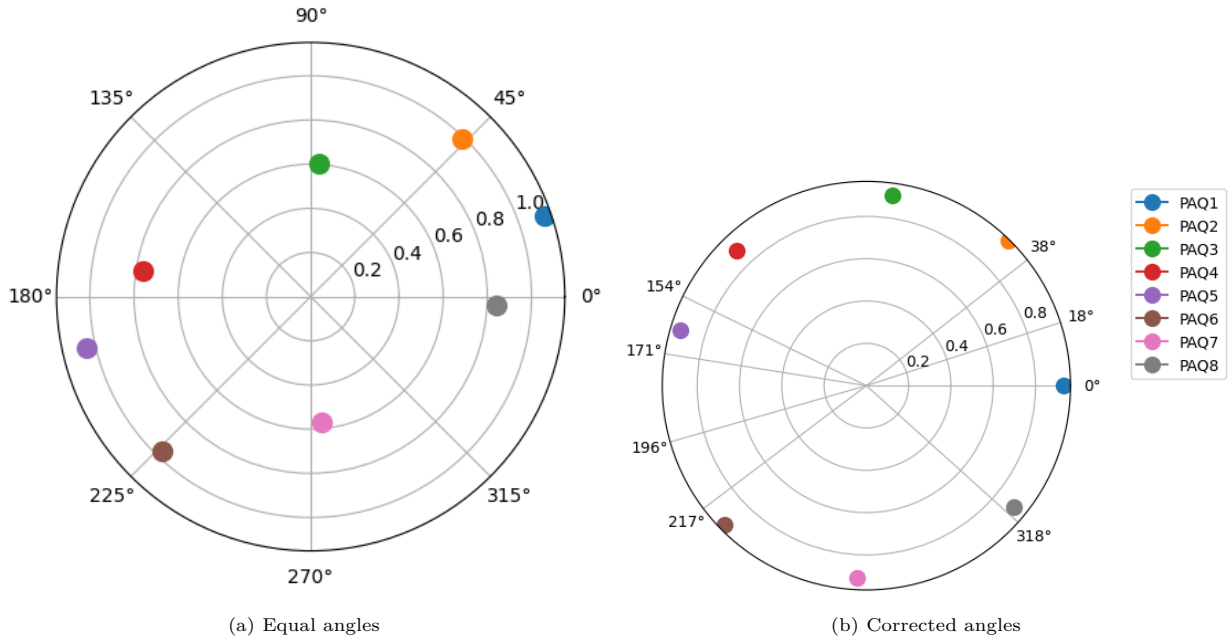


Figure 2: Locating the language-specific circumplex for Mandarin, using equal angles and corrected angles

1.1. Using the corrected angles for ISO 12913-3 and *Mitchell et al. (2022)* style analysis

```

import soundscapy as sspy

def adj_angle_iso_coords(data: pd.DataFrame, angles, scale=100):
    isopl = data.apply(lambda x: adj_iso_pl(x[scales].values, angles, scale=scale), axis=1)
    isoev = data.apply(lambda x: adj_iso_ev(x[scales].values, angles, scale=scale), axis=1)
    return isopl, isoev

def adj_iso_pl(values, angles, scale=None):
    # scale = range of input values (e.g. 0-100)
    # The scaling factor was derived by comparing to the scaling from the ISO method. Confirmed to be
    # 100 * sum of abs values of the loading factors / 2
    iso_pl = np.sum([np.cos(np.deg2rad(angle)) * values[i] for i, angle in enumerate(angles)])
    if scale:
        iso_pl = iso_pl / (scale/2 * np.sum(np.abs([np.cos(np.deg2rad(angle)) for angle in angles])))
    return iso_pl

def adj_iso_ev(values, angles, scale=None):
    iso_ev = np.sum([np.sin(np.deg2rad(angle)) * values[i] for i, angle in enumerate(angles)])
    if scale:

```

```

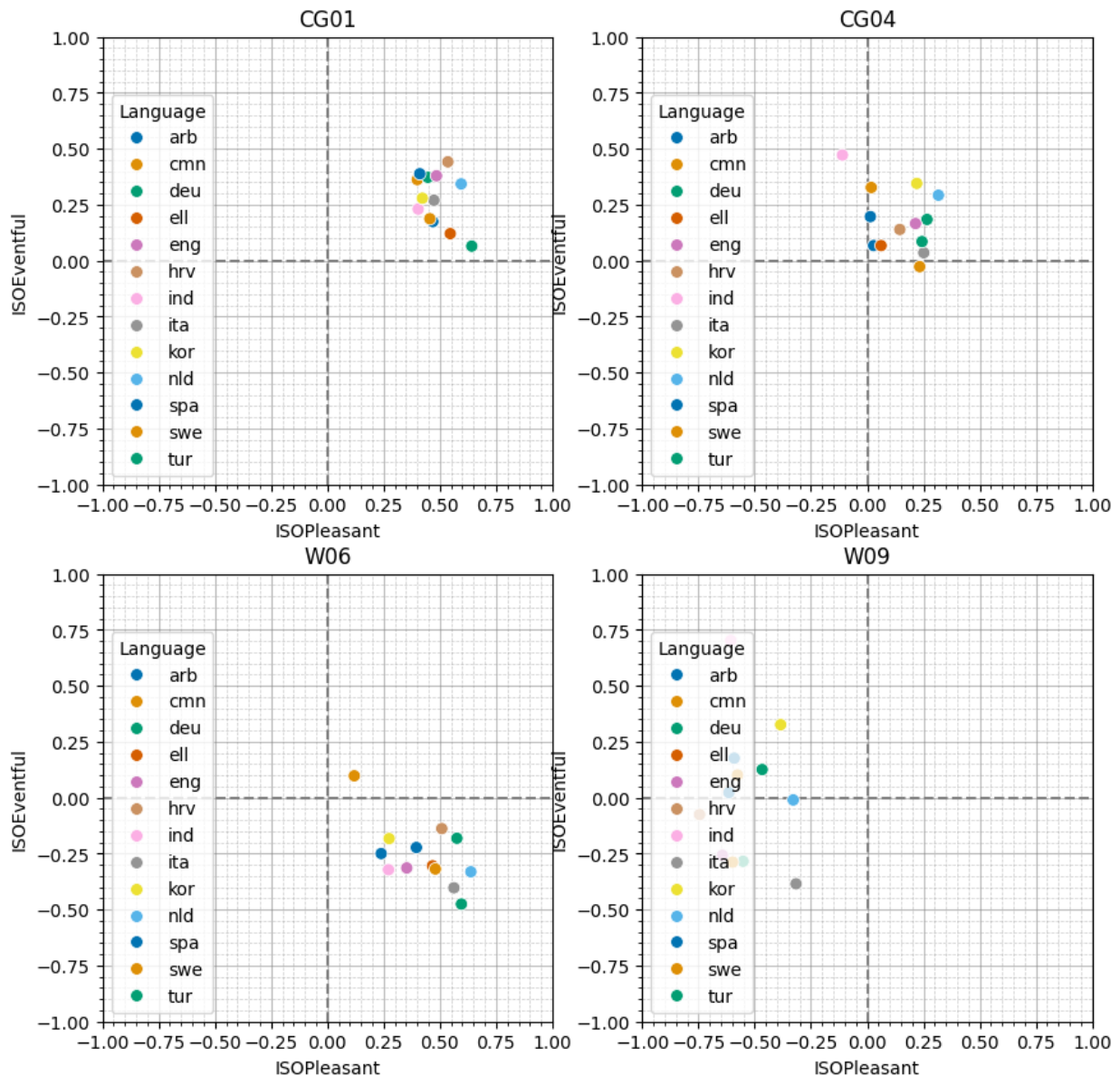
        iso_ev = iso_ev / (scale/2 * np.sum(np.abs([np.sin(np.deg2rad(angle)) for angle in angles])))
    return iso_ev

```

```

fig, axes = plt.subplots(2, 2, figsize=(10, 10))
for i, rec in enumerate(['CG01', 'CG04', 'W06', 'W09']):
    cg01 = satp.query("Recording == @rec")
    cg01_res = {}
    for lang in cg01.Language.unique():
        cg01_lang = cg01.query("Language == @lang")
        isopl, isoev = adj_angle_iso_coords(cg01_lang, angles=ang_dict[lang])
        cg01_res[lang] = {'Language': lang, 'ISOPleasant': np.mean(isopl), 'ISOEventful': np.mean(isoev)}
        # print(np.mean(isopl), np.mean(isoev))
    cg01_res = pd.DataFrame.from_dict(cg01_res, orient='index')
    ssply.plotting.scatter(cg01_res, hue='Language', s=50, title=rec, ax=axes.flatten()[i])

```



```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
cg04 = satp.query("Recording == 'W06'")

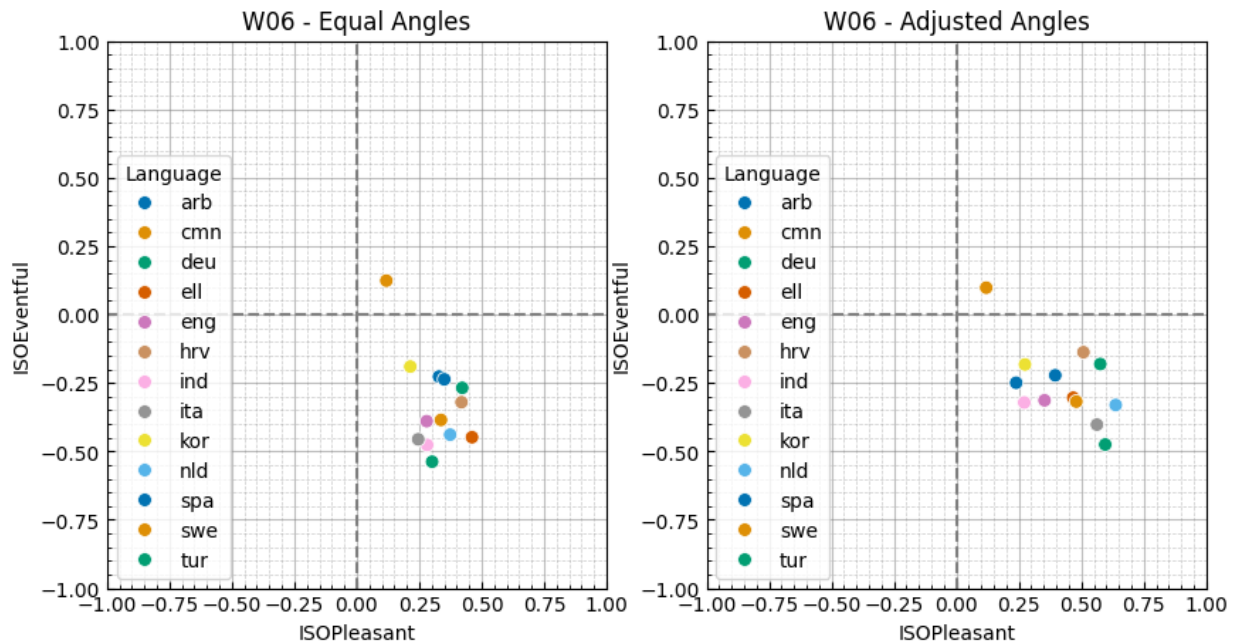
cg04_res = {}
for lang in cg04.Language.unique():
    cg04_lang = cg04.query("Language == @lang")
    isopl, isoev = adj_angle_iso_coords(cg04_lang, angles=eq_angles)
    cg04_res[lang] = {'Language': lang, 'ISOPleasant': np.mean(isopl), 'ISOEventful': np.mean(isoev)}
    # print(np.mean(isopl), np.mean(isoev))
cg04_res = pd.DataFrame.from_dict(cg04_res, orient='index')
sspy.plotting.scatter(cg04_res, hue='Language', s=50, title='W06 - Equal Angles', ax=axes.flatten()[0])

cg04_res = {}
```

```

for lang in cg04.Language.unique():
    cg04_lang = cg04.query("Language == @lang")
    isopl, isoev = adj_angle_iso_coords(cg04_lang, angles=ang_dict[lang])
    cg04_res[lang] = {'Language': lang, 'ISOPleasant': np.mean(isopl), 'ISOEventful': np.mean(isoev)}
    # print(np.mean(isopl), np.mean(isoev))
cg04_res = pd.DataFrame.from_dict(cg04_res, orient='index')
sspy.plotting.scatter(cg04_res, hue='Language', s=50, title='W06 - Adjusted Angles', ax=axes.flatten())

```

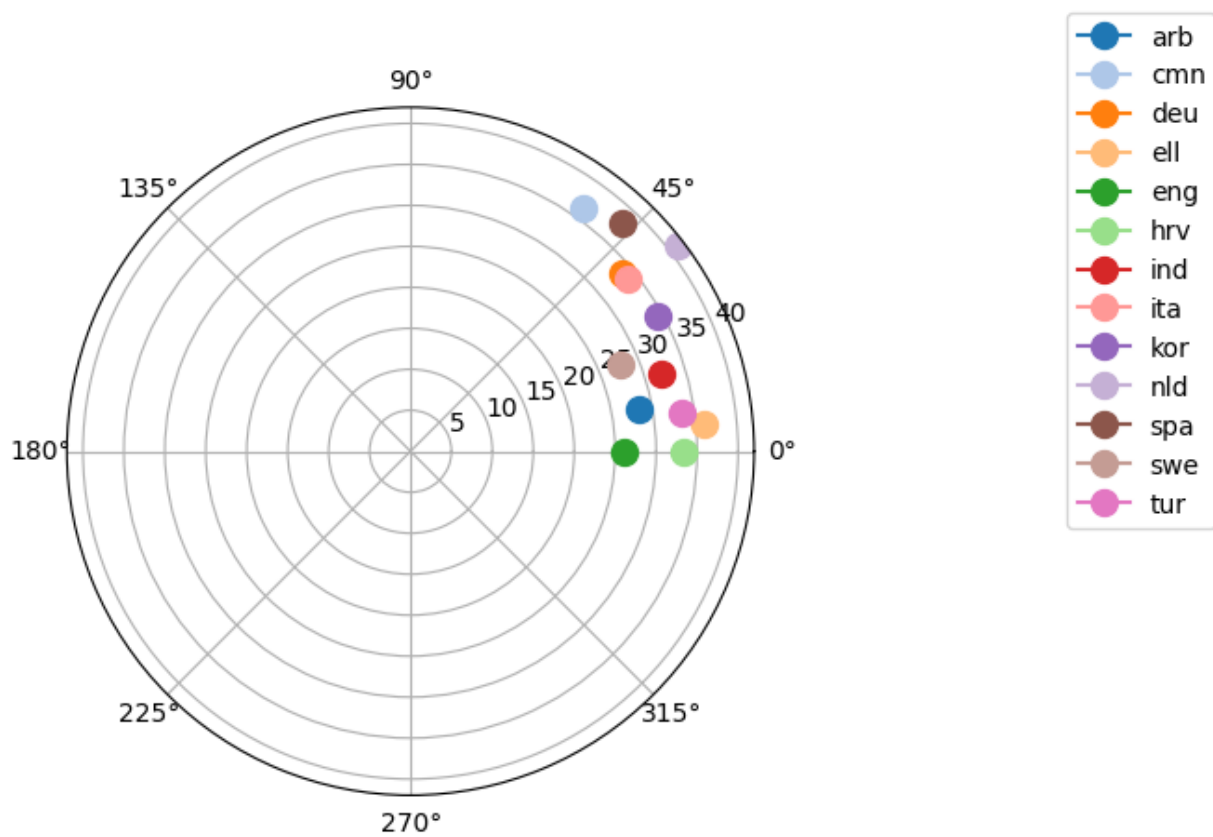


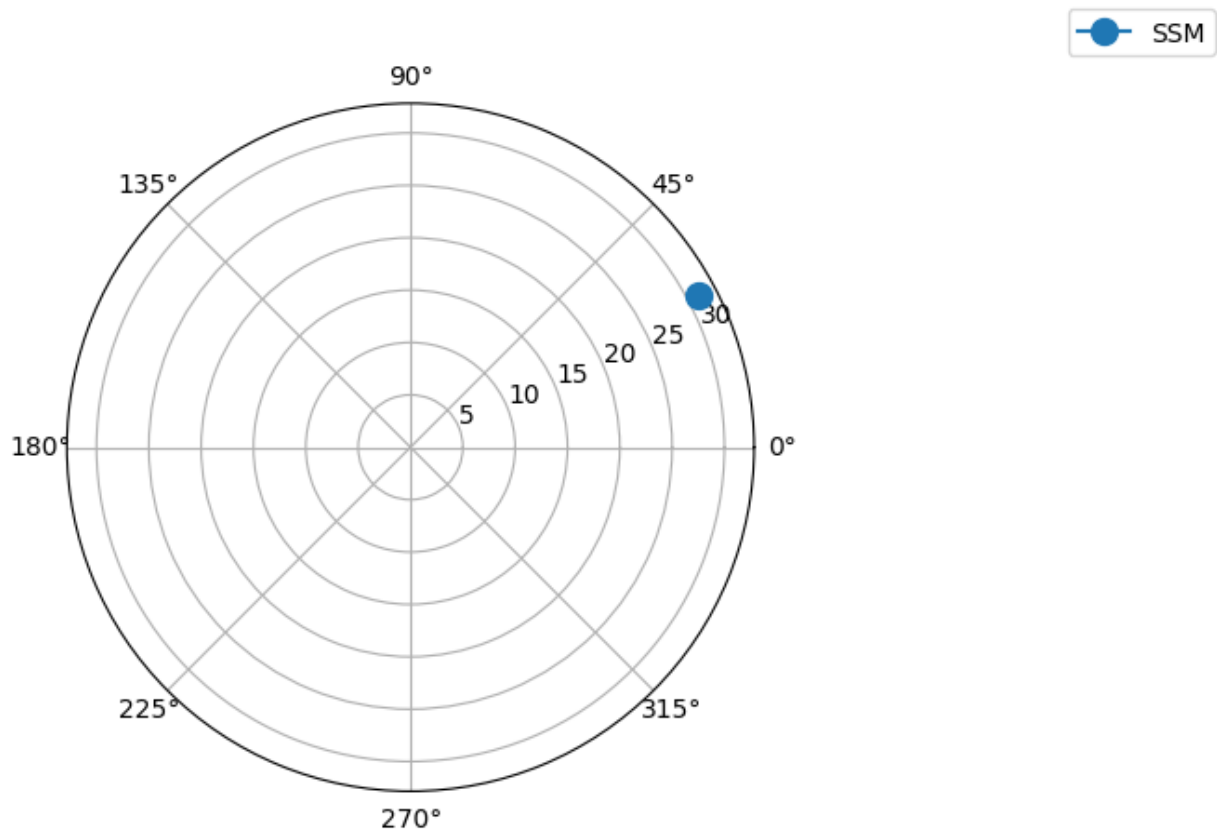
```

cg01_res = circumplex.ssm_analyse(satp.query("Recording == 'CG01'"), scales, grouping=['Language'], g
cg01_all = circumplex.ssm_analyse(satp.query("Recording == 'CG01'"), scales)

cg01_res.plot()
plt.show()
cg01_all.plot()
plt.show()

```





References

- Andreella, A., De Santis, R., Vesely, A., Finos, L., 2023. Procrustes-based distances for exploring between-matrices similarity. *Statistical Methods and Applications* 32, 867–882. doi:[10.1007/s10260-023-00689-y](https://doi.org/10.1007/s10260-023-00689-y).
- Bakhtiar, T., Siswadi, S., 2015. On the symmetrical property of procrustes measure of distance. *International Journal of Pure and Applied Mathematics* 99. doi:[10.12732/ijpam.v99i3.7](https://doi.org/10.12732/ijpam.v99i3.7).
- Mitchell, A., Aletta, F., Kang, J., 2022. How to analyse and represent quantitative soundscape data. *JASA Express Letters* 2, 037201. doi:[10.1121/10.0009794](https://doi.org/10.1121/10.0009794), [arXiv:https://doi.org/10.1121/10.0009794](https://arxiv.org/abs/https://doi.org/10.1121/10.0009794).
- Rogoza, R., Ciecuch, J., Strus, W., 2021. A three-step procedure for analysis of circumplex models: An example of narcissism located within the circumplex of personality metatraits. *Personality and Individual Differences* 169, 109775. doi:[10.1016/j.paid.2019.109775](https://doi.org/10.1016/j.paid.2019.109775).