PCA Preprocessing:

This notebook produces some graphs for the preprocessing section of the report

In [1]:

```
%matplotlib inline
%load_ext autoreload
%autoreload 2
from pathlib import Path

# Enter the locations of the sample directories
CELLO_PATH  = Path("/home/lukas/BA/philharmonia-samples/cello")
GUITAR_PATH = Path("/home/lukas/BA/philharmonia-samples/guitar")

# Output directories for figures and wavfiles
GFX_PATH    = Path("/home/lukas/BA/report/gfx/")
WAVS_PATH   = Path("/home/lukas/BA/report/wavs/")

# Whether to generate graphs for all samples. Image files will be
# written to the dataset directories.
GENERATE_ALL_GRAPHS = True
```

In [2]:

```
# Initialization
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import librosa
import pya
import random

import principal_harmonics as ph

for path in [GFX_PATH, WAVS_PATH]:
    if path.exists() and not path.is_dir():
        raise NotADirectoryError(path)
    if not path.exists():
        path.mkdir()
```

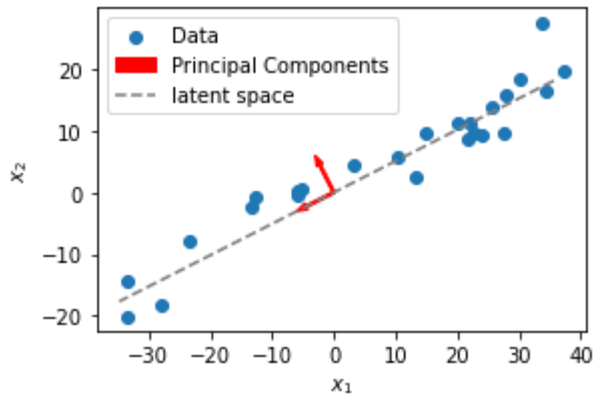
PCA Illustration

The following plot illustrates PCA with artificial 2D data:

In [3]:

```
plt.figure(figsize=(5, 3))

ax = plt.gca()
ax.set_aspect('equal')
rng = np.random.default_rng(seed=42)
u  = np.array([2.0, 1.0])
uo = np.array([-1.0, 2.0])
random_alphas = rng.uniform(-20, 20,       size=(25, 1))
noise         = rng.normal(loc=0, scale=2, size=(25, 1))
pts = u.reshape((1, -1)) * random_alphas + uo * noise

plt.scatter(pts[:, 0], pts[:, 1], label='Data')
pca = PCA()
pca.fit(pts)

u_est  = pca.components_[0]
alphas = np.arange(-40, 40).reshape((-1, 1))
axis   = u_est.reshape((1, -1)) * alphas
#plt.quiver([ 0], [ 0], pca.components_[0,0], pca.components_[0,1], units='xy', scale=0.1)
plt.arrow(0, 0, *5*pca.components_[0], width=0.4, facecolor='red', edgecolor='red', label='Principal Components')
plt.arrow(0, 0, *5*pca.components_[1], width=0.4, facecolor='red', edgecolor='red')
plt.plot(axis[:, 0], axis[:, 1], c='gray', linestyle='--', label='latent space')
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")

plt.legend()
plt.savefig(GFX_PATH / '2-pca-illustration.eps')
plt.tight_layout()
```

```
The PostScript backend does not support transparency; partially transparent artists will be rendered opaque.
```



Sub-Noise Amplitude Imputation for a Guitar Sample

Using iterative imputation (unstable)

Iterative imputation allows to preserve more detail.

In [4]:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from principal_harmonics.models import *

guitar_path = GUITAR_PATH / 'guitar_D5_very-long_forte_normal.mp3'
guitar_asig = pya.Asig(str(guitar_path)).norm()
guitar_pitch = ph.pvoc.constant_pitch(ph.pvoc.get_pitch(guitar_asig), librosa.note_to_hz('D5'))

guitar_freqs, guitar_coefs = ph.pvoc.sinusoidal_analysis(guitar_asig, guitar_pitch,
                                                         interpolate_hole_limit=10,
                                                         remove_short_limit=5,
                                                         peak_matching='simple', n_periods=6)

clipper = ph.pvoc.ClipTransient()
start, end = clipper.clip(guitar_coefs)
guitar_freqs = guitar_freqs[start:end]; guitar_coefs = guitar_coefs[start:end]
guitar_ampls = np.abs(guitar_coefs)

imputer_pipeline = make_pipeline(
    StandardScaler(),
    Ridge()
)
pipeline = make_pipeline(
    DropDCTransformer(),
    DBTransformer(),
    HoleImputer(hole_size_limit=5),
    IterativeSubNoiseImputer(constant_value=-240, estimator=imputer_pipeline, initial_strategy='constant',
                             max_value=-80, max_iter=20, random_state=42, use_time=True),
)
guitar_ampls_not_imputed          = pipeline[:-1].fit_transform(guitar_ampls)
guitar_ampls_iteratively_imputed  = pipeline.fit_transform(guitar_ampls)
guitar_ampls_iteratively_imputed[~np.isnan(guitar_ampls_not_imputed)] = np.nan
```

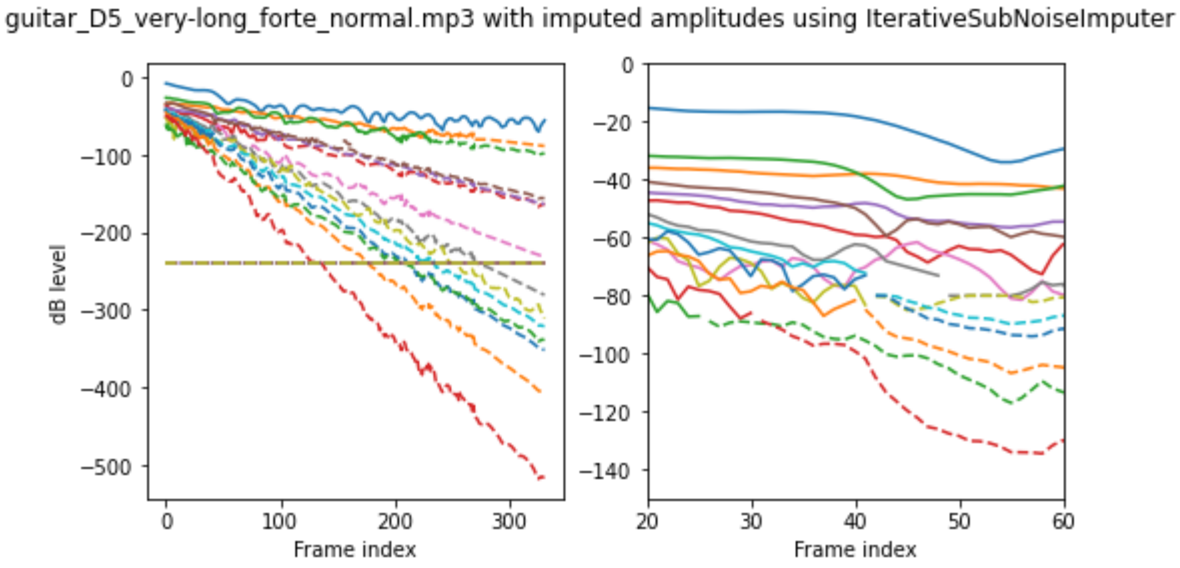
```
/home/lukas/.local/lib/python3.9/site-packages/sklearn/impute/_iterative.py:685: ConvergenceWarning: [IterativeImpu
ter] Early stopping criterion not reached.
  warnings.warn("[IterativeImputer] Early stopping criterion not"
```

In [5]:

```
def plot_subnoise(ampls_not_imputed, ampls_imputed):
    ax1 = plt.subplot(121)
    plt.ylabel("dB level")
    plt.xlabel("Frame index")
    T, n  = ampls_imputed.shape
    for i in range(n):
        plt.plot(ampls_not_imputed[:, i],           c='C'+str(i%10))
        plt.plot(ampls_imputed[:, i], linestyle='dashed', c='C'+str(i%10))

    plt.subplot(122)
    for i in range(n):
        plt.plot(ampls_not_imputed[:, i],           c='C'+str(i%10))
        plt.plot(ampls_imputed[:, i], linestyle='dashed', c='C'+str(i%10))

    plt.xlim(20, 60)
    plt.ylim(-150, 0)

    plt.xlabel("Frame index")

plt.figure(figsize=(8.2, 4))
plot_subnoise(guitar_ampls_not_imputed, guitar_ampls_iteratively_imputed)
plt.suptitle(f"{guitar_path.name} with imputed amplitudes using IterativeSubNoiseImputer")
```

Out[5]:

```
Text(0.5, 0.98, 'guitar_D5_very-long_forte_normal.mp3 with imputed amplitudes using IterativeSubNoiseImputer')
```

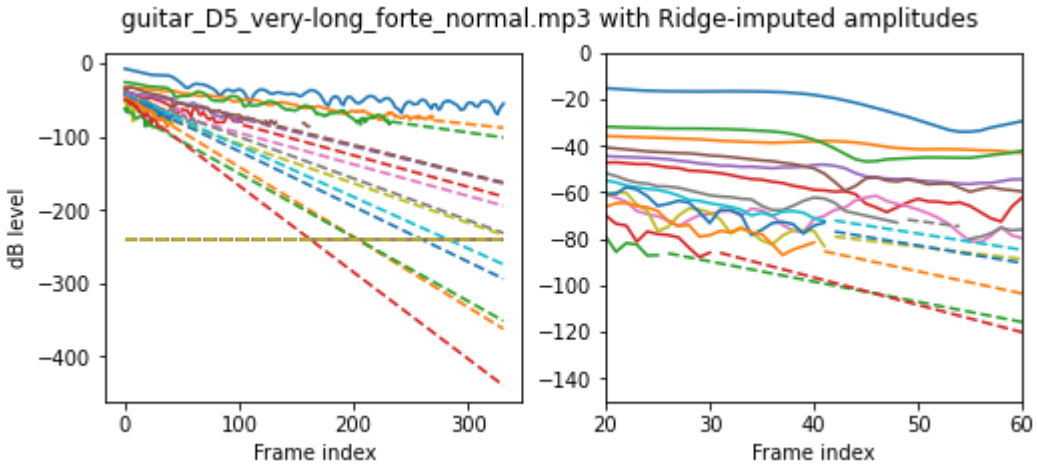


It is not clear if this level of detail is actually justified. Instead:

Using Ridge imputation (less detailed, but stable)

In [6]:
```
pipeline = make_pipeline(
    DropDCTransformer(),
    DBTransformer(),
    HoleImputer(hole_size_limit=5),
    RidgeSubNoiseImputer(constant_value=-240)
)
guitar_ampls_not_imputed   = pipeline[:-1].fit_transform(guitar_ampls)
guitar_ampls_ridge_imputed = pipeline.fit_transform(guitar_ampls)
guitar_ampls_ridge_imputed[~np.isnan(guitar_ampls_not_imputed)] = np.nan
```

In [7]:
```
plt.figure(figsize=(8.2, 3.2))
plot_subnoise(guitar_ampls_not_imputed, guitar_ampls_ridge_imputed)
plt.suptitle(f"{guitar_path.name} with Ridge-imputed amplitudes")
plt.savefig(GFX_PATH / "2-sub-noise-imputation.eps")
```


guitar_D5_very-long_forte_normal.mp3 with Ridge-imputed amplitudes
dB level
Frame index
Frame index