Python音声処理入門（第3部）

音声合成・音声認識・高度な音声処理

第3部の内容

- テキスト読み上げ（TTS: Text-to-Speech）の2つのアプローチ
- 音声認識（STT: Speech-to-Text）とデータ変換
- 音声フォーマット変換の実践技術
- PyWorldによる音声分析・合成理論
- ボイスチェンジャーの数学的実装

6. テキスト読み上げ（TTS）の技術比較

オフライン vs オンライン

オフライン（pyopenjtalk）

✓ 超高速処理（リアルタイム倍速以上）
✓ ネット不要、プライバシー保護
✓ 複数話者対応（.htsvoiceファイル）
✗ 音質は中程度、日本語のみ

オンライン（gTTS）

✓ 最高品質、非常に自然な音声
✓ 多言語対応（100言語以上）
✗ ネット必須、API制限あり
✗ 遅延が大きい、プライバシー懸念

practice_5_tts.py : Open JTalk 詳細解説

Open JTalkの音声合成プロセス

```python
import pyopenjtalk

# 1段階目: テキスト解析
text = "こんにちは、音声合成の世界へようこそ。"

# 内部的に実行される処理:
# 1. 形態素解析（mecab-naist-jdic）
# 2. 音韻変換（ひらがな → 音素列）
# 3. アクセント・韻律解析
# 4. HMM音声合成

# 2段階目: 音声生成
x, sr = pyopenjtalk.tts(text)

print(f"生成された音声:")
print(f"  サンプリング周波数: {sr} Hz")
print(f"  データ長: {len(x)} サンプル")
print(f"  時間: {len(x)/sr:.2f} 秒")
print(f"  データ型: {x.dtype}")
```

Open JTalkの詳細パラメータ制御

```python
import pyopenjtalk

def advanced_tts(text, speed=1.0, pitch=0.0, alpha=0.55):
    """
    高度なパラメータ制御でのTTS
    Args:
        text: 読み上げテキスト
        speed: 話速倍率（0.5=半分速度, 2.0=2倍速度）
        pitch: ピッチシフト（+0.1=高く, -0.1=低く）
        alpha: 声質パラメータ（0.0-1.0, 0.55がデフォルト）
    """
    # フルオプションでの合成
    x, sr = pyopenjtalk.tts(
        text,
        speed=speed,
        pitch=pitch,
        alpha=alpha
    )

    return x, sr

# 使用例
normal_voice, sr = advanced_tts("通常の声です。")
fast_voice, sr = advanced_tts("早口で話します。", speed=1.5)
high_pitch_voice, sr = advanced_tts("高い声で話します。", pitch=0.2)
robot_voice, sr = advanced_tts("ロボット風の声です。", alpha=0.1)
```

Open JTalkの話者（ボイス）切り替え

.htsvoiceファイルの活用

```
import pyopenjtalk
import glob

# 利用可能な話者ファイルを探索
voice_files = glob.glob("voices/*.htsvoice")
print("利用可能な話者:")
for voice in voice_files:
    print(f"  {voice}")

def tts_with_voice(text, voice_file=None):
    """
    指定した話者で音声合成
    """
    if voice_file:
        # 話者ファイルを指定して合成
        x, sr = pyopenjtalk.tts(text, voice=voice_file)
    else:
        # デフォルト話者で合成
        x, sr = pyopenjtalk.tts(text)

    return x, sr

# 異なる話者での合成例
text = "話者を変更してみます。"
default_voice, sr = tts_with_voice(text)
custom_voice, sr = tts_with_voice(text, "voices/mei_normal.htsvoice")
```

practice_8_gtts.py : Google TTS詳細解説

gTTSの内部処理フロー

```
from gtts import gTTS
import pygame
import io
import os

def google_tts_detailed(text, lang='ja', slow=False, tld='com'):
    """
    Google TTSの詳細制御
    Args:
        text: 読み上げテキスト
        lang: 言語コード（'ja', 'en', 'ko', 'zh'等）
        slow: 低速再生フラグ
        tld: トップレベルドメイン（'com', 'co.jp'等で音質が微妙に変化）
    """
    # Google TTSオブジェクト作成
    tts = gTTS(
        text=text,
        lang=lang,
        slow=slow,
        tld=tld  # 'com', 'co.jp', 'co.uk' など
    )
    # メモリ上に音声データを保存
    audio_buffer = io.BytesIO()
    tts.write_to_fp(audio_buffer)
    audio_buffer.seek(0)
    return audio_buffer
```

Google TTSの多言語対応

各言語での音声合成例

```
# 多言語対応例
texts = [
    ("こんにちは、日本語の音声合成です。", 'ja'),
    ("Hello, this is English text-to-speech.", 'en'),
    ("안녕하세요, 한국어 음성 합성입니다.", 'ko'),
    ("你好，这是中文语音合成。", 'zh')
]

for text, lang in texts:
    audio_data = google_tts_detailed(text, lang=lang)
    # 再生処理...
```

対応言語例

日本語(ja), 英語(en), 韓国語(ko), 中国語(zh), フランス語(fr), ドイツ語(de), スペイン語(es), イタリア語(it), ロシア語(ru), アラビア語(ar) など100言語以上

Google TTSの実践的活用

```python
import pygame
import requests
from gtts import gTTS
import tempfile
import os

def play_gtts_audio(text, lang='ja', cleanup=True):
    """
    Google TTS音声の生成と再生（自動クリーンアップ付き）
    """
    # 一時ファイルの作成
    with tempfile.NamedTemporaryFile(suffix='.mp3', delete=False) as tmp_file:
        temp_filename = tmp_file.name

    try:
        # 音声合成
        tts = gTTS(text=text, lang=lang)
        tts.save(temp_filename)

        # pygame初期化（まだの場合）
        if not pygame.mixer.get_init():
            pygame.mixer.init(frequency=22050, size=-16, channels=2, buffer=512)

        # 再生
        pygame.mixer.music.load(temp_filename)
        pygame.mixer.music.play()

        # 再生完了まで待機
        while pygame.mixer.music.get_busy():
            pygame.time.wait(100)

    except requests.exceptions.RequestException as e:
        print(f"ネットワークエラー: {e}")
    except Exception as e:
        print(f"音声合成エラー: {e}")
    finally:
        # クリーンアップ
        if cleanup and os.path.exists(temp_filename):
            os.remove(temp_filename)
```

7. 音声認識（STT）の理論と実装

Voskの技術的背景

Kaldi音声認識ツールキットをベースとした高精度エンジン

技術的特徴

- **深層学習（DNN）+ 隠れマルコフモデル（HMM）**
- **完全オフライン、リアルタイム処理対応**
- **軽量（モデルサイズ: 50MB～500MB）**
- **ストリーミング認識対応**

言語モデルの構成

```
Vosk日本語モデル（ja-0.22）の内部構造:
├── am/          # 音響モデル（音素→音声の対応）
├── graph/       # 発音辞書・言語モデル
├── ivector/     # 話者適応用特徴量
└── conf/        # 設定ファイル
```

practice_6_stt.py の詳細実装

```python
import sounddevice as sd
import numpy as np
from vosk import Model, KaldiRecognizer
import json
import os

# Voskに最適化されたパラメータ
FS = 16000  # Voskの推奨サンプリング周波数
DURATION = 5
CHANNELS = 1  # モノラルのみ対応

def setup_vosk_model(model_path="models/vosk/ja-0.22"):
    """
    Voskモデルの初期化と検証
    """
    if not os.path.exists(model_path):
        raise FileNotFoundError(f"Voskモデルが見つかりません: {model_path}")

    # モデル読み込み（メモリ大量消費するため1回のみ）
    print("Voskモデルを読み込み中...")
    model = Model(model_path)

    # 認識器作成
    recognizer = KaldiRecognizer(model, FS)

    # 部分認識結果も取得する設定
    recognizer.SetWords(True)  # 単語レベルの時間情報

    print("音声認識エンジン準備完了")
    return model, recognizer
```

データ変換の詳細メカニズム

sounddevice → Vosk変換プロセス

```python
def convert_audio_for_vosk(recording):
    """
    sounddevice出力をVosk入力形式に変換

    変換ステップ:
    1. float32 [-1.0, 1.0] → int16 [-32768, 32767]
    2. numpy配列 → bytes列
    """

    print(f"変換前: dtype={recording.dtype}, shape={recording.shape}")
    print(f"値の範囲: [{recording.min():.3f}, {recording.max():.3f}]")

    # Step 1: float32 → int16変換
    # クリッピング対策を含む変換
    clipped = np.clip(recording, -1.0, 1.0)  # 範囲制限
    scaled = clipped * 32767  # [-32767, 32767]の範囲にスケール
    int16_data = scaled.astype(np.int16)

    print(f"変換後: dtype={int16_data.dtype}")
    print(f"値の範囲: [{int16_data.min()}, {int16_data.max()}]")

    # Step 2: bytes形式に変換
    bytes_data = int16_data.tobytes()
    print(f"バイト列長: {len(bytes_data)} bytes")

    return bytes_data
```

リアルタイム音声認識の実装

```
def streaming_speech_recognition(duration=10):
    """
    リアルタイムストリーミング音声認識
    """
    model, recognizer = setup_vosk_model()

    def audio_callback(indata, frames, time, status):
        """
        音声入力コールバック関数
        """
        if status:
            print(f"オーディオエラー: {status}")

        # データ変換
        audio_bytes = convert_audio_for_vosk(indata[:, 0])  # モノラル抽出

        # 音声認識実行
        if recognizer.AcceptWaveform(audio_bytes):
            # 文が完了した場合
            result = json.loads(recognizer.Result())
            if result['text']:
                print(f"認識結果: {result['text']}")
        else:
            # 部分認識結果
            partial = json.loads(recognizer.PartialResult())
            if partial['partial']:
                print(f"認識中: {partial['partial']}", end='¥r')

    # ストリーミング開始
    print(f"{duration}秒間のリアルタイム認識を開始...")
    with sd.InputStream(callback=audio_callback,
                        channels=1,
                        samplerate=FS,
                        blocksize=4000):  # 0.25秒分のデータを一度に処理
        sd.sleep(duration * 1000)  # ミリ秒指定
```

音声認識の精度向上テクニック

```python
def preprocess_audio_for_stt(audio, sample_rate):
    """
    音声認識精度向上のための前処理
    """
    from scipy import signal

    # 1. ノイズゲート（無音部分の雑音除去）
    noise_threshold = 0.02
    audio_gated = np.where(np.abs(audio) > noise_threshold, audio, 0)

    # 2. 高域通過フィルタ（低周波雑音除去）
    sos = signal.butter(4, 80, btype='high', fs=sample_rate, output='sos')
    audio_filtered = signal.sosfilt(sos, audio_gated)

    # 3. 正規化（一定の音量レベルに調整）
    max_val = np.max(np.abs(audio_filtered))
    if max_val > 0:
        audio_normalized = audio_filtered * (0.7 / max_val)
    else:
        audio_normalized = audio_filtered

    # 4. 16kHzにリサンプリング（必要に応じて）
    if sample_rate != 16000:
        audio_resampled = signal.resample(
            audio_normalized,
            int(len(audio_normalized) * 16000 / sample_rate)
        )
    else:
        audio_resampled = audio_normalized

    return audio_resampled.astype(np.float32)
```

8. 音声フォーマット変換：ffmpeg完全活用

ffmpegの技術的背景

最強のマルチメディア変換ツール

圧倒的な対応範囲

- **対応コーデック**: 数百種類
- **対応フォーマット**: 音声・動画合わせて1000種類以上
- **処理機能**: 変換、フィルタ、エフェクト、結合、分割
- **パフォーマンス**: GPU加速、並列処理対応

Pythonからの安全なffmpeg実行

```python
import subprocess
import shlex
import os

def safe_ffmpeg_command(input_file, output_file, options=[]):
    """
    安全なffmpegコマンド実行
    """
    # 基本コマンド構築
    cmd = ['ffmpeg', '-i', input_file] + options + ['-y', output_file]

    # コマンド表示（デバッグ用）
    print(f"実行コマンド: {' '.join(cmd)}")

    try:
        # 標準出力・エラーをキャプチャして実行
        result = subprocess.run(
            cmd,
            check=True,                 # エラー時に例外発生
            capture_output=True,        # 出力をキャプチャ
            text=True,                  # 文字列として扱う
            timeout=300                 # 5分でタイムアウト
        )

        print(f"変換成功: {input_file} → {output_file}")
        return True

    except subprocess.CalledProcessError as e:
        print(f"ffmpegエラー: {e}")
        print(f"標準エラー: {e.stderr}")
        return False
    except subprocess.TimeoutExpired:
        print("ffmpeg処理がタイムアウトしました")
        return False
```

practice_7_converter.py 高度な実装

```python
def convert_wav_to_mp3(input_wav, output_mp3, bitrate="192k", quality=2):
    """
    WAV → MP3変換（高品質設定）

    Args:
        bitrate: ビットレート（'128k', '192k', '320k'）
        quality: 品質設定（0=最高品質～9=最低品質）
    """
    options = [
        '-c:a', 'libmp3lame',    # MP3エンコーダー指定
        '-b:a', bitrate,         # ビットレート
        '-q:a', str(quality),    # 品質パラメータ
        '-joint_stereo', '1',    # ジョイントステレオ
        '-lowpass', '20000'      # ローパスフィルタ（20kHz）
    ]

    return safe_ffmpeg_command(input_wav, output_mp3, options)

def convert_to_multiple_formats(input_file, base_name):
    """
    1つの音声ファイルを複数フォーマットに一括変換
    """
    conversions = [
        # (拡張子, オプション, 説明)
        ('.mp3', ['-c:a', 'libmp3lame', '-b:a', '192k'], 'MP3 192kbps'),
        ('.ogg', ['-c:a', 'libvorbis', '-q:a', '6'], 'OGG Vorbis 高品質'),
        ('.m4a', ['-c:a', 'aac', '-b:a', '128k'], 'AAC 128kbps'),
        ('.flac', ['-c:a', 'flac'], 'FLAC 可逆圧縮'),
    ]

    results = {}
    for ext, options, desc in conversions:
        output_file = f"{base_name}{ext}"
        print(f"変換中: {desc}")
        success = safe_ffmpeg_command(input_file, output_file, options)
        results[ext] = success

    return results
```

音声エフェクト付き変換

```python
def apply_audio_effects_with_ffmpeg(input_file, output_file, effects):
    """
    ffmpegのオーディオフィルタを使用したエフェクト処理
    """
    filter_parts = []

    # 音量調整
    if 'volume' in effects:
        filter_parts.append(f"volume={effects['volume']}")

    # イコライザー（低音、中音、高音）
    if 'equalizer' in effects:
        eq = effects['equalizer']
        if 'bass' in eq:
            filter_parts.append(f"bass=g={eq['bass']}")
        if 'treble' in eq:
            filter_parts.append(f"treble=g={eq['treble']}")

    # リバーブ（エコー）
    if 'reverb' in effects:
        reverb_params = effects['reverb']
        delay = reverb_params.get('delay', 0.5)
        decay = reverb_params.get('decay', 0.5)
        filter_parts.append(f"aecho={decay}:{decay}:{int(delay*1000)}:{decay}")

    # ノイズゲート
    if 'noise_gate' in effects:
        threshold = effects['noise_gate']
        filter_parts.append(f"gate=threshold={threshold}:ratio=1:attack=3:release=10")

    # フィルタチェーン構築
    if filter_parts:
        filter_chain = ','.join(filter_parts)
        options = ['-af', filter_chain]
    else:
        options = []

    return safe_ffmpeg_command(input_file, output_file, options)

# 使用例：ラジオボイス風エフェクト
radio_voice_effects = {
    'volume': 1.2,          # 音量20%アップ
    'equalizer': {
        'bass': -5,         # 低音-5dB
        'treble': 3         # 高音+3dB
    },
    'noise_gate': -40,      # -40dB以下をカット
}

apply_audio_effects_with_ffmpeg(
    "input.wav",
    "radio_voice.mp3",
    radio_voice_effects
)
```

9. PyWorldによる音声分析・合成理論

人間の音声生成メカニズム

音声 = 音源 + フィルタ

音源フィルタモデル

音源成分:
- 有声音: 声帯の周期振動 → 基本周波数（F0）
- 無声音: 乱流による雑音 → 非周期性成分

フィルタ成分:
- 声道（口、舌、喉の形状）→ スペクトル包絡
- 共鳴特性によって倍音を強調/減衰

PyWorldの3要素分解

PyWorld要素の音響的意味

F0（基本周波数）
声の高さ、話者の性別・年齢を決定
男性: 100-200Hz
女性: 200-400Hz

SP（スペクトル包絡）
声色、母音の識別、話者性
声道の共鳴特性を表現
個人の声質を決定

AP（非周期性指標）
息遣い、ささやき声、摩擦音
0.0 = 完全周期（機械的）
1.0 = 完全雑音（息音）

practice_9_pyworld_voicemod.py 詳細実装

```python
import pyworld as pw
import numpy as np
import soundfile as sf
import sounddevice as sd

def analyze_voice_with_pyworld(audio, fs):
    """
    PyWorldによる詳細な音声分析
    """
    # データ型をfloat64に変換（PyWorld要求）
    if audio.dtype != np.float64:
        audio = audio.astype(np.float64)

    print("PyWorld音声分析開始...")

    # 1. 基本周波数抽出（DIO + StoneMask）
    print("  F0抽出中...")
    f0, t = pw.dio(audio, fs)          # 高速だが粗い
    f0_refined = pw.stonemask(audio, f0, t, fs)  # 精密化

    # 2. スペクトル包絡抽出
    print("  スペクトル包絡抽出中...")
    sp = pw.cheaptrick(audio, f0_refined, t, fs)

    # 3. 非周期性指標抽出
    print("  非周期性指標抽出中...")
    ap = pw.d4c(audio, f0_refined, t, fs)

    print("分析完了!")
    print(f"  F0形状: {f0_refined.shape} (時間フレーム数)")
    print(f"  SP形状: {sp.shape} (時間 × 周波数ビン)")
    print(f"  AP形状: {ap.shape} (時間 × 周波数ビン)")

    return f0_refined, sp, ap, t
```

音声分析結果の統計処理

```python
def analyze_voice_statistics(f0, sp, ap, fs):
    """
    音声分析結果の詳細統計
    """
    # F0統計（有声部分のみ）
    voiced_frames = f0 > 0
    if np.any(voiced_frames):
        f0_mean = np.mean(f0[voiced_frames])
        f0_std = np.std(f0[voiced_frames])
        f0_min = np.min(f0[voiced_frames])
        f0_max = np.max(f0[voiced_frames])

        print(f"F0統計:")
        print(f"  平均F0: {f0_mean:.1f} Hz")
        print(f"  標準偏差: {f0_std:.1f} Hz")
        print(f"  範囲: {f0_min:.1f} - {f0_max:.1f} Hz")

        # 話者性別推定
        if f0_mean < 165:
            gender_guess = "男性"
        else:
            gender_guess = "女性"
        print(f"  推定性別: {gender_guess}")

    # 非周期性統計
    ap_mean = np.mean(ap)
    print(f"非周期性統計:")
    print(f"  平均AP: {ap_mean:.3f} (0=機械的, 1=息音的)")

    # スペクトル統計
    sp_power = np.mean(sp, axis=0)  # 時間平均したスペクトル
    print(f"スペクトル統計:")
    print(f"  周波数ビン数: {len(sp_power)}")

    return {
        'f0_mean': f0_mean if np.any(voiced_frames) else 0,
        'f0_std': f0_std if np.any(voiced_frames) else 0,
        'ap_mean': ap_mean,
        'gender_guess': gender_guess if np.any(voiced_frames) else "不明"
    }
```

高度なボイスチェンジング技術

```python
def pitch_shift_advanced(f0, semitones):
    """
    音楽的ピッチシフト（半音単位）

    Args:
        f0: 基本周波数配列
        semitones: 半音数（+12 = 1オクターブ上, -12 = 1オクターブ下）
    """
    # 半音 = 2^(1/12)倍
    ratio = 2 ** (semitones / 12.0)

    # 有声部分のみ変更
    voiced_mask = f0 > 0
    modified_f0 = f0.copy()
    modified_f0[voiced_mask] *= ratio

    print(f"ピッチシフト: {semitones:+.1f}半音 (倍率: {ratio:.3f})")
    return modified_f0

def formant_shift(sp, fs, shift_ratio=1.0):
    """
    フォルマント（声道特性）シフト

    Args:
        sp: スペクトル包絡
        shift_ratio: シフト倍率（>1.0で高く, <1.0で低く）
    """
    if shift_ratio == 1.0:
        return sp

    modified_sp = np.zeros_like(sp)
    freq_bins = sp.shape[1]

    for t in range(sp.shape[0]):
        for f in range(freq_bins):
            # 周波数をシフト
            shifted_f = int(f * shift_ratio)
            if 0 <= shifted_f < freq_bins:
                modified_sp[t, shifted_f] = sp[t, f]

    print(f"フォルマントシフト: {shift_ratio:.3f}倍")
    return modified_sp
```

性別変換の数学的実装

```python
def gender_conversion(f0, sp, fs, target_gender='female'):
    """
    性別変換（統計的手法）
    """
    if target_gender == 'female':
        # 女性化: ピッチ上昇 + フォルマント上昇
        f0_modified = pitch_shift_advanced(f0, +5)      # +5半音
        sp_modified = formant_shift(sp, fs, 1.15)       # フォルマント15%上昇
        print("女性化処理完了")
    else:  # male
        # 男性化: ピッチ下降 + フォルマント下降
        f0_modified = pitch_shift_advanced(f0, -3)      # -3半音
        sp_modified = formant_shift(sp, fs, 0.9)        # フォルマント10%下降
        print("男性化処理完了")

    return f0_modified, sp_modified

def synthesize_voice_with_pyworld(f0, sp, ap, fs):
    """
    PyWorldによる音声再合成
    """
    print("PyWorld音声合成中...")
    synthesized = pw.synthesize(f0, sp, ap, fs)

    # クリッピング防止
    max_val = np.max(np.abs(synthesized))
    if max_val > 1.0:
        synthesized = synthesized / max_val * 0.95
        print(f"  正規化: {max_val:.3f} → 0.95")

    return synthesized.astype(np.float32)
```

特殊音声効果の実装

```python
def robot_voice_effect(f0, sp, ap, fs, robot_f0=150):
    """
    ロボット声エフェクト
    """
    # F0を一定値に固定
    robot_f0_array = np.full_like(f0, robot_f0)
    # 無声部分は0のまま
    robot_f0_array[f0 == 0] = 0

    # 非周期性を減少（機械的な音にする）
    mechanical_ap = ap * 0.3

    print(f"ロボット声: F0固定={robot_f0}Hz")
    return robot_f0_array, sp, mechanical_ap

def whisper_effect(f0, sp, ap, fs):
    """
    ささやき声エフェクト
    """
    # F0をほぼゼロに（無声化）
    whisper_f0 = f0 * 0.1

    # 非周期性を大幅増加
    whisper_ap = np.minimum(ap * 5.0, 0.95)  # 上限0.95

    # 高周波成分を強調
    enhanced_sp = sp.copy()
    # 高周波数帯域（後半）を強調
    high_freq_start = sp.shape[1] // 2
    enhanced_sp[:, high_freq_start:] *= 1.5

    print("ささやき声エフェクト適用")
    return whisper_f0, enhanced_sp, whisper_ap
```

周波数領域エフェクト

```python
def echo_in_frequency_domain(sp, delay_frames=10, decay=0.6):
    """
    周波数領域でのエコー処理
    """
    echo_sp = sp.copy()

    # 遅延フレーム分だけずらして加算
    if delay_frames < sp.shape[0]:
        echo_sp[delay_frames:] += sp[:-delay_frames] * decay

    print(f"周波数エコー: {delay_frames}フレーム遅延, 減衰{decay}")
    return echo_sp

def frequency_masking_effect(sp, mask_start=0.3, mask_end=0.7, mask_strength=0.1):
    """
    特定周波数帯域のマスキング（フィルタ効果）
    """
    masked_sp = sp.copy()
    freq_bins = sp.shape[1]

    start_bin = int(freq_bins * mask_start)
    end_bin = int(freq_bins * mask_end)

    # 指定範囲の周波数を減衰
    masked_sp[:, start_bin:end_bin] *= mask_strength

    print(f"周波数マスキング: {mask_start*100:.0f}%-{mask_end*100:.0f}%帯域を{mask_strength}倍に減衰")
    return masked_sp
```

完全なボイスチェンジャーシステム

```python
def complete_voice_changer(input_file, output_file, effect_type, **kwargs):
    """
    完全なボイスチェンジャーシステム

    Args:
        effect_type: 'pitch', 'gender', 'robot', 'whisper', 'echo'
        **kwargs: エフェクト固有のパラメータ
    """
    # 1. 音声読み込み
    print(f"音声ファイル読み込み: {input_file}")
    x, fs = sf.read(input_file, dtype=np.float64)
    if x.ndim == 2:
        x = x.mean(axis=1)  # モノラル化

    # 2. PyWorld分析
    f0, sp, ap, t = analyze_voice_with_pyworld(x, fs)

    # 3. エフェクト適用
    if effect_type == 'pitch':
        semitones = kwargs.get('semitones', 0)
        f0_mod = pitch_shift_advanced(f0, semitones)
        sp_mod, ap_mod = sp, ap

    elif effect_type == 'gender':
        target = kwargs.get('target_gender', 'female')
        f0_mod, sp_mod = gender_conversion(f0, sp, fs, target)
        ap_mod = ap

    elif effect_type == 'robot':
        robot_f0 = kwargs.get('robot_f0', 150)
        f0_mod, sp_mod, ap_mod = robot_voice_effect(f0, sp, ap, fs, robot_f0)

    elif effect_type == 'whisper':
        f0_mod, sp_mod, ap_mod = whisper_effect(f0, sp, ap, fs)

    elif effect_type == 'echo':
        delay = kwargs.get('delay_frames', 10)
        decay = kwargs.get('decay', 0.6)
        f0_mod, ap_mod = f0, ap
        sp_mod = echo_in_frequency_domain(sp, delay, decay)

    else:
        raise ValueError(f"未対応エフェクト: {effect_type}")

    # 4. 再合成
    synthesized = synthesize_voice_with_pyworld(f0_mod, sp_mod, ap_mod, fs)

    # 5. 保存
    sf.write(output_file, synthesized, fs)
    print(f"処理完了: {output_file}")

    return synthesized, fs
```

実用的な使用例

コマンドライン実行例

```
# 標準的なピッチ変更
python practice_9_pyworld_voicemod.py

# ファイルとパラメータを詳細指定
python practice_9_pyworld_voicemod.py ¥
    -i my_voice.wav ¥
    -o modified_voice.wav ¥
    --f0_rate 1.5 ¥
    --effect pitch
```

プログラムからの呼び出し例

```
# ピッチを5半音上げる
result, fs = complete_voice_changer(
    "input.wav",
    "high_pitch.wav",
    "pitch",
    semitones=5
)

# 女性化
result, fs = complete_voice_changer(
    "male_voice.wav",
    "female_voice.wav",
```

バッチ処理とワークフロー

複数ファイルの一括処理

```
# 複数ファイルに同じ処理を適用
for file in *.wav; do
    python practice_9_pyworld_voicemod.py ¥
        -i "$file" ¥
        -o "processed_${file}" ¥
        --f0_rate 0.8
done
```

処理チェーンの構築

```
def audio_processing_chain(input_file, output_dir):
    """
    複数エフェクトの連続適用
    """
    base_name = os.path.splitext(os.path.basename(input_file))[0]

    # 各エフェクトを順次適用
    effects = [
        ('pitch', {'semitones': 3}),
        ('gender', {'target_gender': 'female'}),
        ('robot', {'robot_f0': 140}),
        ('whisper', {}),
    ]
```

第3部まとめ

習得した高度な技術

音声合成（TTS）
オフライン・オンライン両手法の特徴理解と実装

音声認識（STT）
データ変換からリアルタイム処理まで完全習得

フォーマット変換
ffmpegの実践的活用とエフェクト処理

音声分析・変調
PyWorldによる人間音声の数学的理解

技術的深掘りポイント

- **データ型変換**: float32 ↔️ int16、精度と互換性の理解
- **リアルタイム処理**: コールバック関数とストリーミング技術
- **信号処理理論**: 周波数領域操作、フィルタリング技術
- **音響学基礎**: 音源フィルタモデル、フォルマント理論の実装

総合まとめ：Python音声処理の全体像

技術スタックの階層構造

```
低レベル:   Numpy配列、数値計算
中レベル:   sounddevice、soundfile、scipy
高レベル:   AI系ライブラリ（Vosk、PyWorld）
応用レベル: 独自アルゴリズム、リアルタイム処理
```

実用アプリケーション例

1. **音声アシスタント**: STT + 自然言語処理 + TTS
2. **音声コンテンツ制作**: エフェクト + フォーマット変換
3. **アクセシビリティツール**: 読み上げ + 音声認識
4. **エンターテイメント**: ボイスチェンジャー + ゲーム連携

次のステップへの道筋

さらなる発展への道

Python音声処理の基礎から応用まで完全習得！

次に挑戦すべき技術領域

- **深層学習**: TensorFlow/PyTorchとの連携
- **リアルタイム処理**: より低遅延なシステム設計
- **Web統合**: Flask/Djangoでのオンラインサービス
- **組み込み応用**: Raspberry Pi等での実装
- **音声AI**: ChatGPT Voice、Whisper等最新技術

継続学習のポイント

- **実践重視**: 小さなプロジェクトから始める
- **コミュニティ参加**: オープンソースプロジェクトへの貢献
- **最新情報**: 音声処理の研究動向をフォロー
- **応用展開**: 自分の専門分野との融合を模索

おめでとうございます！

Python音声処理の包括的な知識と技術を習得されました

これからは実践で力を発揮してください！