Lucas Moura Veloso

Sintetizador de voz para a avaliação da qualidade da voz disfônica

Belo Horizonte

Junho de 2017

Lucas Moura Veloso

Sintetizador de voz para a avaliação da qualidade da voz disfônica

Monografia apresentada durante o Seminário dos Trabalhos de Conclusão do Curso de Graduação em Engenharia Elétrica da UFMG, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista. A área de concentração é a de processamento digital de voz.

Universidade Federal de Minas Gerais – UFMG

Escola de Engenharia

Curso de Graduação em Engenharia Elétrica

Orientador: Prof. Dr. Maurílio Nunes Vieira

Belo Horizonte

Junho de 2017

Este trabalho é dedicado a todas as pessoas que em algum momento pensaram que não eram boas o suficiente para serem quem queriam.

Agradecimentos

Primeiramente, gostaria de agradecer à minha família e amigos, que me aguentaram durante essa época particularmente estressante da minha vida, não teria conseguido terminar sem o apoio de vocês. Kenia, Léo, Carolina, Camila, Marinalva, Guido, Thaís, Cris, Matheus, Sanja, Afonso, Carol, Wagner, Thays e mais um monte de gente, há espaço limitado nessa página.

Gostaria de agradecer também ao meu orientador, Prof. Dr. Maurílio Nunes Vieira, que foi mais do que prestativo durante essa jornada e que me mostrou o caminho para essa área do conhecimento, que pouco tinha ouvido falar até então.

Outra pessoa que merece agradecimentos é o Prof. Dr. Hani Camille Yehia, que me apresentou esta área durante uma das suas disciplinas e que me ensinou a usar o LaTeX.

"There's nothing that can't be done, if we raise our voice as one."
(Michael Jackson, We've had enough)

Resumo

Este trabalho trata do desenvolvimento de uma aplicação Python para profissionais de fonoaudiologia, com ferramentas de síntese de voz específicas para a avaliação da qualidade de vozes disfônicas. A topologia do sintetizador utilizado neste trabalho é baseada no modelo fonte-filtro da produção da voz e na síntese por formantes. **Palavras-chaves**: síntese, voz, filtros, digitais.

Lista de ilustrações

Lista de tabelas

Lista de símbolos

$F0$	Frequência fundamental do pulso glótico
FG	Frequência do pulso glótico
k	Fator de assimetria do pulso glótico
AN	Amplitude do formante nasal
$A1$	Amplitude do primeiro formante
FNP	Frequência do polo nasal
BNP	Largura de banda do polo nasal
BNZ	Largura de banda do zero nasal
$F1$	Frequência do primeiro formante
$B1$	Largura de banda do primeiro formante
$F2$	Frequência do segundo formante
$B2$	Largura de banda do segundo formante
$F3$	Frequência do terceiro formante
$B3$	Largura de banda do terceiro formante
$B4$	Largura de banda do quarto formante
$B5$	Largura de banda do quinto formante
z	Operador de deslocamento da transformada Z
BW	Largura de banda
F	Frequência de corte
T	Período de amostragem

Sumário

1 Introdução

Atualmente, o diagnóstico fonoaudiológico de distúrbios da voz é dado de forma subjetiva e requer extremo treinamento da audição do profissional e conta com a sua habilidade de discernimento de certas características da voz. O objetivo deste trabalho é criar uma ferramenta computacional que facilite este trabalho, sintetizando vozes programaticamente com perturbações que simulam os diversos tipos de vozes que um paciente poderia apresentar, reduzindo assim a quantidade de treinamento necessário e o elemento do erro humano.

Para isso será desenvolvido uma variação do sintetizador de formantes descrito Klatt (1980) com algumas mudanças baseadas na metodologia descrita em Fant (1979) e em metodologias desenvolvidas pelo aluno e seu orientador, que permita não apenas uma síntese de voz, como também a adição de diversos tipos de controles e funcionalidades que simulem as diversas disfonias da voz, de forma similar aos resultados descritos em Lucero et al. (2014).

Este trabalho está organizado da seguinte forma:

a) Introdução;

b) Revisão bibliográfica;

c) Teoria;

d) Metodologia;

e) Desenvolvimento e Resultados;

f) Trabalhos futuros;

g) Conclusão;

h) Bibliografia;

i) Apêndices.

2 Revisão Bibliográfica

Neste capítulo iremos discutir um pouco do histórico do problema, as soluções aceitas na literatura para o mesmo e o atual estado da arte.

O sintetizador de formantes de Klatt (1980) foi escolhido para este problema devido à sua ampla aceitação e revisão no meio acadêmico, mas a síntese por formantes não é a única ferramenta que poderia ter sido escolhida. Nas subseções abaixo falaremos um pouco sobre os outros tipos de síntese de fala que poderiam ter sido utilizadas e finalmente sobre a ferramenta escolhida.

2.1 História dos sintetizadores de voz

Talvez a ideia mais antiga de síntese de voz seja a síntese por articulação, sendo que suas origens podem ser rastreadas para os séculos 11 e 12. De forma muito arcaica estas máquinas são descritas como "mechanical talking heads", eram idealizadas como objetos físicos que simulavam os processos do trato vocal. Apesar de sua origem antiga, a primeira aparição na literatura científica formal destes aparelhos se deu em Tarnóczy (1949).

No começo do século 20, os avanços na Engenharia Elétrica permitiram o desenvolvimento de sintetizadores de voz eletrônicos. O primeiro destes dispositivos a atingir um grande público foi o VODER, criado por Homer Dudley, porém este precisava de muito treinamento para ser utilizado, como explicado em Traunmüller (2005).

Durante os anos seguintes a área de sintese de voz continuou a crescer seguindo um caminho mais funcional e menos lúdico e baseado no entretenimento, foi então que começaram a surgir os sintetizadores analógicos que moldaram o caminho para o estado da arte atual. Como exemplo importante dos sintetizadores analógicos podemos destacar

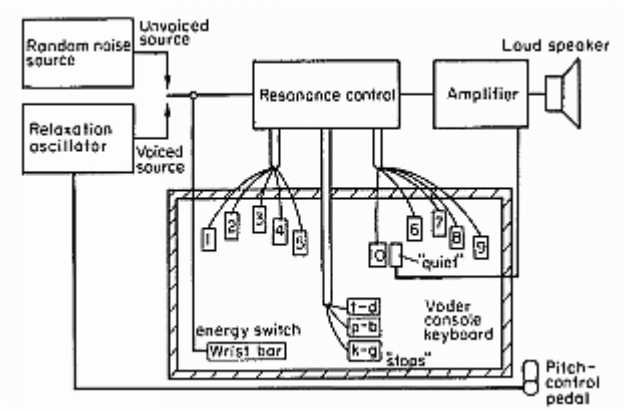

Figura 1 – Diagrama de funcionamento do VODER. Fonte: Traunmüller (2005).

Figura 2 – Pessoa operando o VODER na *World Fair* de Nova York em 1939. Fonte: Traunmüller (2005).

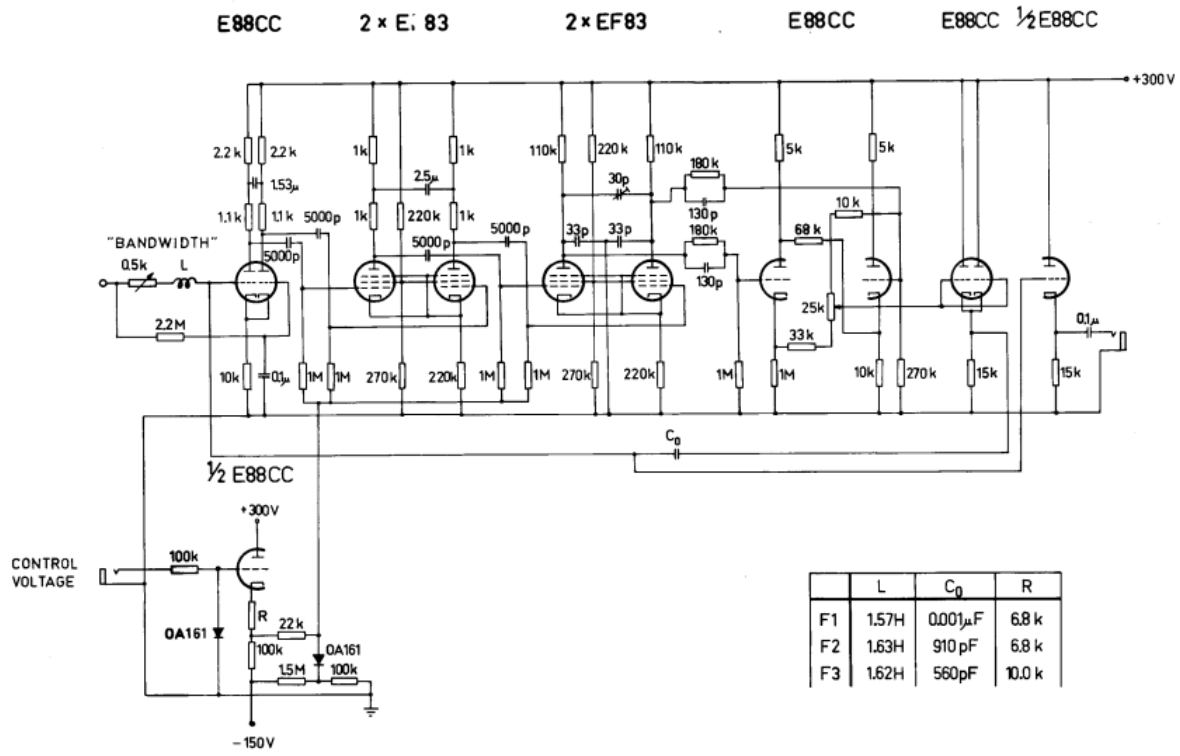

Figura 3 – Diagrama do circuito analógico para gerar os filtros ressoadores de formantes controlados por tensão do OVE II. Fonte: Fant e Martony (1962).

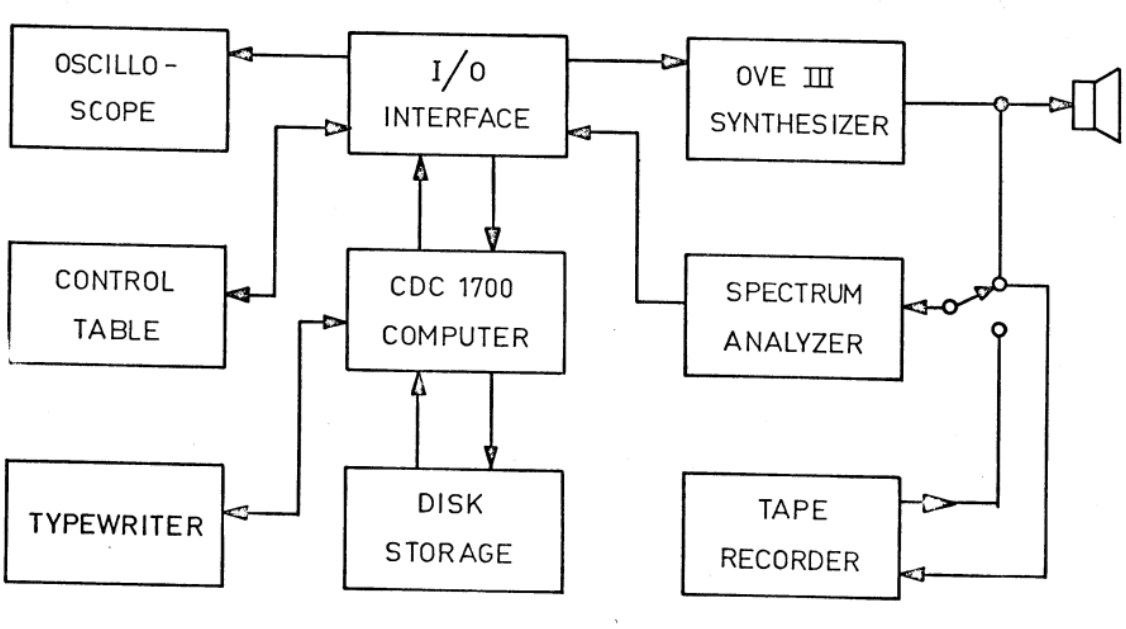

Figura 4 – Diagrama geral do OVE III. Fonte: Liljencrants (1968).

o OVE (TRAUNMÜLLER, 2005) e suas versões posteriores OVE II (FANT; MARTONY, 1962) e OVE III (LILJENCRANTS, 1968) criados por Fant, G., Liljencrants, J. e a equipe do *Royal Institute of Technology in Stockholm*.

Um aspecto interessante das versões do OVE desenvolvidas por Fant (I e II) é que eram totalmente analógicas e controladas manualmente, fazendo uso de circuitos analógicos para gerar e tratar os sinais de voz. Já o OVE III começou a usar circuitos digitais para algumas operações como uso básico de um teclado e uso de registradores e unidades de computação. As figuras 3 e 4 demonstram um pouco do funcionamento destes modelos.

2.2 Estado da arte

2.2.1 Síntese por articulação

Esta técnica, descrita brevemente na seção 2.1, atualmente é aplicável em software, com os mesmos conceitos de tentar emular os processos de formação de voz através de simulação dos processos físicos envolvidos.

2.2.2 Síntese por concatenação

Este método de síntese é baseada na junção de partes de áudios de fala para montar novas palavras. Existem vários tipo de implementações para esta arquitetura, porém a sua maior desvantagem é a necessidade de uma grande base de dados de amostras de fala e

uma certa mecanicidade perceptível na fala devido aos cortes nos áudios, o que pode ser reduzido com o uso de algoritmos adequados de concatenação.

Uma implementação para este método pode ser encontrada em Hunt e Black (1996) e um estudo sobre percepção humana e medidas objetivas foi realizado em Stylianou e Syrdal (2001).

2.2.3 Síntese por formantes

A síntese por formantes não usa de áudios pré-gravados e nem tenta simular os processos físicos por trás da formação da fala, ao invés disso, o objetivo é gerar sinais e tratá-los com técnicas de processamento de sinais, de forma que o resultado final se aproxime o máximo possível da voz humana.

A sua base foi originalmente descrita em Gold e Rabiner (1968) e sua implementação foi feita inicialmente em Klatt (1980), onde descreve uma arquitetura de sintetizador que utiliza as formantes da voz e diversos outros parâmetros de controle para sintetizar vozes como vogais e fricativas. Esta configuração é amplamente aceita na comunidade acadêmica e já foi revisada extensamente, fazendo desta uma ótima plataforma para o desenvolvimento de um sintetizador de vozes disfônicas.

2.2.4 Síntese de vozes disfônicas

O campo de síntese de vozes disfônicas não trata de uma arquitetura de sintetizador e nem de técnicas para os mesmos, mas sim de uma visão mais geral da síntese digital de voz, que leva em consideração as disfunções do trato vocal que uma pessoa pode ter e sua síntese, gerando assim uma voz que se aproxime mais da real do locutor.

Atualmente esta é uma área muito estudada, alguns trabalhos que podem ser citados como referência do atual estado da arte nacional são Lucero et al. (2014), que descreve um sintetizador digital por articulação de vozes disfônicas, e Vieira, Sansao e Yehia (2014) que neste trabalho mediram o ruído glotal de vozes humanas e sintéticas por meio de algoritmos de processamento de imagens 2D.

3 Teoria

Neste capítulo descreveveremos a teoria por trás da síntese por formantes para entendimento do trabalho. Maiores informações podem ser encontradas em Klatt (1980).

3.1 Modelo fonte-filtro da produção da voz

O modelo fonte-filtro da produção da voz, do modo como foi utilizado neste trabalho, basicamente visa emular o comportamento da laringe, faringe e das cavidades oral e nasal ao emitir sons, a partir de técnicas de processamento de sinais.

O pulmão gera um fluxo de ar, que ao passar pelas pregas vocais e em seguida pelas cavidades oral e nasal, são refinados e modulados, gerando assim a voz humana como conhecemos. A figura 6 mostra o processo de abertura e fechamento da glote e como esta manipula o fluxo de ar que vem dos pulmões e gera os pulsos necessários para a vocalização. As cavidades do trato vocal, juntamente com a movimentação dos seus músculos, são os elementos de interesse deste modelo.

As fontes do modelo são equivalentes aos pulsos gerados, como demonstrado na figura 6, enquanto os filtros são relacionados com os formantes e as características de radiação da onda sonora ao sair da cavidade oral, que se comporta como um filtro passa-altas. A figura 7 mostra basicamente como a modelagem é feita.

Os formantes são regiões de alta energia do espectro sonoro e a intenção dos filtros é selecionar as frequências da onda gerada pelo pulso glótico para simular estas regiões de energia elevada. Segundo Vieira (1999), as ordens dos formantes são relevante para síntese da voz da seguinte maneira:

Figura 5 – Diagrama esquemático do trato vocal. Fonte: Vieira (1999).

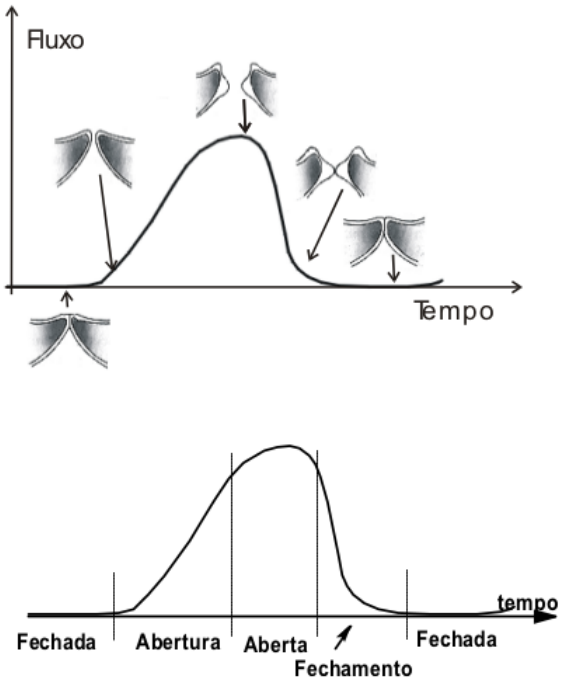

Figura 6 – Diagrama de como a glote manipula o fluxo de ar para gerar os pulsos necessários para a vocalização. Fonte: Vieira (2004).

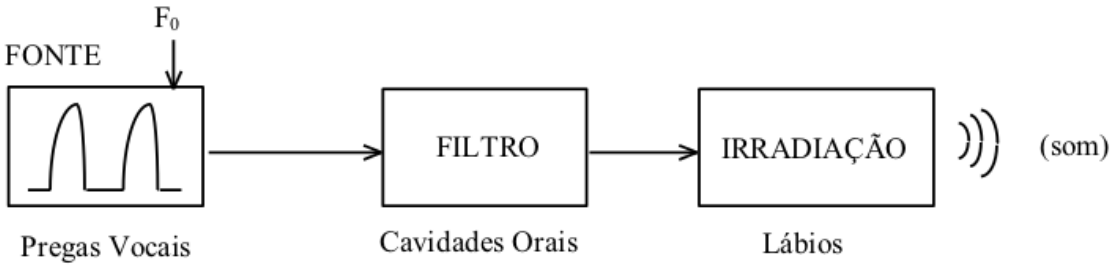

Figura 7 – Diagrama esquemático básico do modelo fonte-filtro da produção da voz. Fonte: Vieira (1999).

> Os três primeiros formantes (F1, F2, F3) estão associados com o tipo de vogal, enquanto os formantes de ordem superior estão mais relacionados com a identidade vocal do locutor. (VIEIRA, 1999).

O filtro de irradiação, ou radiação, figura 7, modela o comportamento da onda sonora ao sair do ambiente pressurizado da cavidade bucal e para o ambiente externo. Este pode ser implementado de diversas formas e será discutido com maior ênfase na seção 3.3.

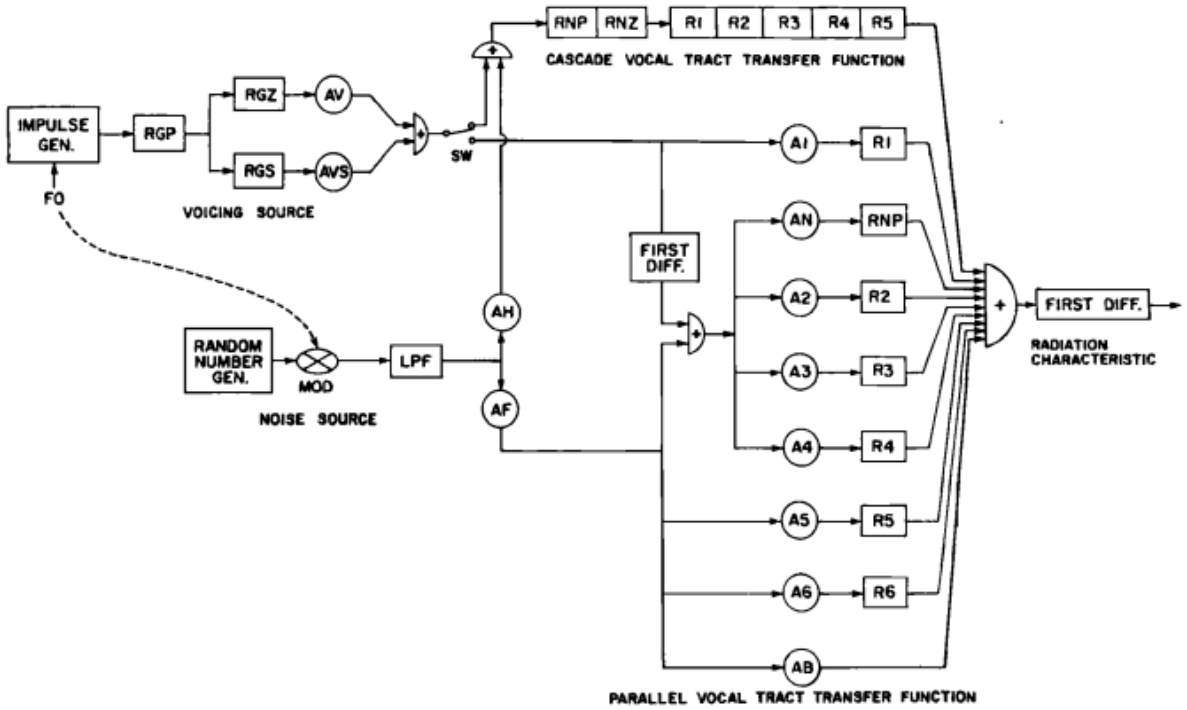

Figura 8 – Diagrama de blocos do sintetizador de formantes. Fonte: Klatt (1980).

Figura 9 – Forma de onda obtida usando a metodologia de Klatt (1980) para o pulso glotal. Fonte: Klatt (1980).

3.2 Teoria geral do sintetizador por formantes

A figura 8 abaixo nos mostra o diagrama de blocos utilizado para geração do sinal de voz. A primeira etapa da síntese é a geração dos sinais de base, sendo estes uma série de pulsos com frequência igual à frequência fundamental escolhida para a fala e uma série aleatória para representar o ruído branco, que será tratado e adicionado para gerar os fricativos da voz.

Os blocos $Ri, 1 \leq i \leq 5$ são filtros passafaixas ressonantes, que, quando parametrizados corretamente, modelam a série de pulsos com frequência fundamental F0 de forma a obter uma forma de onda semelhante aos pulsos glóticos da voz humana. A figura 9 mostra a forma de onda obtida ao aplicarmos a metodologia de geração de pulsos glóticos descrita em Klatt (1980).

A geração dos sinais pode se dar duas formas com o sintetizador de formantes, com

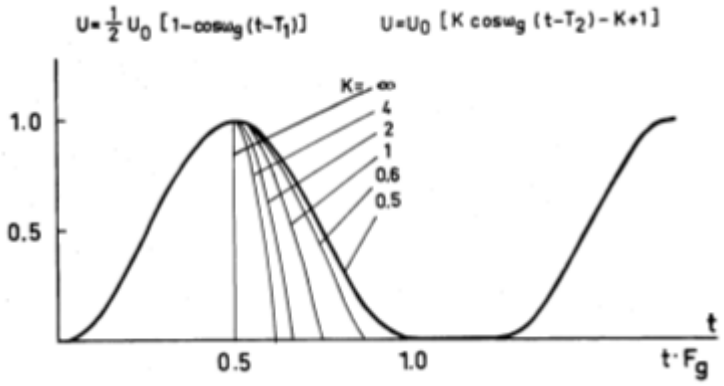

Figura 10 – Modelo de geração dos pulsos glóticos. Fonte: Fant (1979).

a topologia em cascata ou em paralelo. Para este trabalho a implementação escolhida foi a em cascata por ser de implementação mais simples e de manutenção mais fácil, já que elimina a necessidade de ajustar os ganhos dos filtros separadamente.

A abordagem descrita em Klatt (1980) para geração de pulsos glóticos não é a mais adequada para este trabalho, pois existe uma falha básica na sua premissa que é o seu espelhamento no eixo do tempo quando comparado com pulsos reais (ver figuras 6 e 9), além de fornecer pouca capacidade de customização da forma de onda a ser formada. Por estes motivos foi escolhida a abordagem descrita em Fant (1979), onde o pulso glótico é gerado numericamente, baseado na frequência fundamental F0, a frequência do pulso glótico FG e uma constante k que define a descida do pulso.

O sistema foi desenvolvido de forma inteiramente digital mas poderia ser desenvolvido por filtros analógicos conhecidos e estudados durante o curso de Engenharia Elétrica, aplicando o diagrama acima em um sistema analógico, de forma similar aos implementados em Fant e Martony (1962) e Liljencrants (1968).

Neste trabalho, o objetivo é que o sintetizador consiga gerar a vogal /a/. Para conseguir isto, os parâmetros utilizados foram calibrados de acordo com os valores descritos em Klatt (1980), que foram obtidos de vozes reais e trazem maior verossimilhança para este trabalho.

3.3 Filtros digitais ressonantes e antirressonantes

Este trabalho faz uso de diversos tipos de filtros digitais, portanto a teoria por trás destes será brevemente explicada.

Filtros ressonantes possuem uma característica em especial que os separam dos demais. Sua resposta frequência é de tal forma que a frequência de corte e a largura de banda formam uma faixa estreita que filtra qualquer frequência que não seja muito próxima da frequência de corte, que no caso deste trabalho é a frequência da formante.

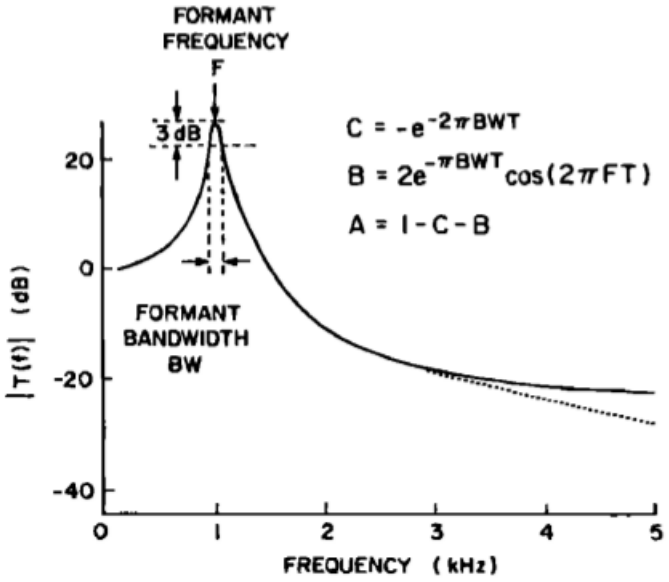

Figura 11 – Resposta em frequência de um filtro digital ressonante de formantes. Fonte: Klatt (1980).

Um filtro deste tipo é exemplificado na figura 11.

Estes filtros podem ser implementados das mais diversas formas e com várias tecnologias, porém utilizaremos apenas suas versões digitais, devido à natureza da implementação do projeto. De acordo com a metodologia descrita em Klatt (1980), temos duas formas de desenvolver estes filtros, uma sequencial e baseada em iterações sobre blocos, ver figura 12, e uma implementação voltada para a função de transferência do geral do filtro ressonante de formantes, que foi a elegida para ser utilizada. Esta equação no domínio da transformada Z é dada por:

$$FiltroFormante(z) = \frac{A}{1 - Bz^{-1} - Cz^{-2}} \tag{3.1}$$

O operador de deslocamento é dado pela variável z e os parâmetros A, B e C dependem de características da voz que queremos sintetizar, como pode ser visto nas equações 3.2, 3.3 e 3.4, respectivamente.

$$A = 1 - C - B \tag{3.2}$$

$$B = 2e^{-\pi \cdot BW \cdot T} \cos(2\pi \cdot F \cdot T) \tag{3.3}$$

$$C = -e^{-2\pi \cdot BW \cdot T} \tag{3.4}$$

Os parâmetros BW, F e T das equações 3.3 e 3.4 representam, respectivamente, a largura de banda desejada para o filtro, a frequência de corte desejada e o período de amostragem utilizado.

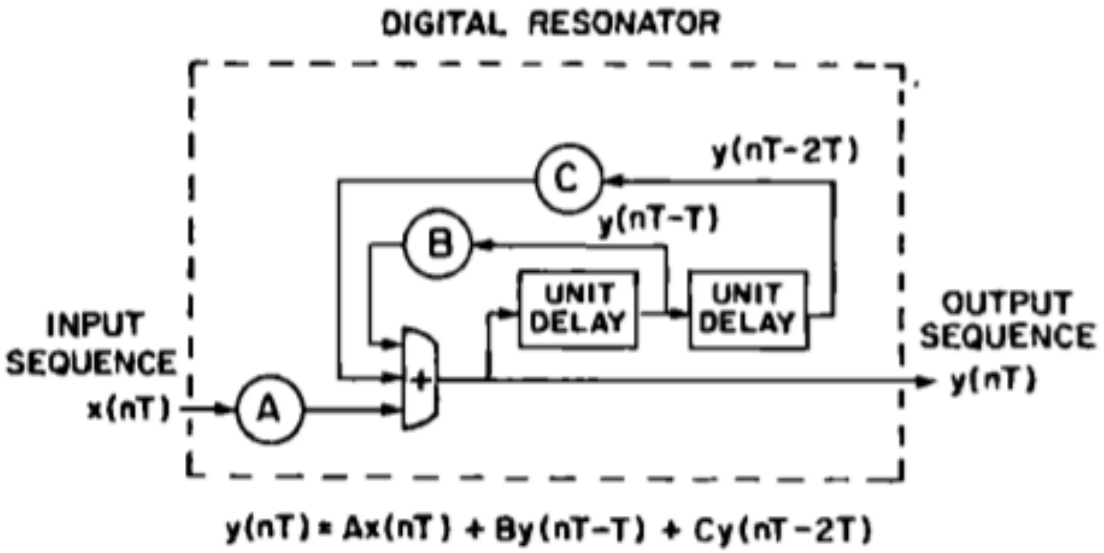

Figura 12 – Diagrama de blocos de um filtro digital ressonante. Fonte: Klatt (1980).

O filtro antirressonante rejeitafaixa que modela o comportamento da cavidade nasal é basicamente o oposto de um filtro ressonante, sendo que sua resposta em frequência é o oposto da de um filtro ressonante baseado nos mesmos parâmetros, sua implementação neste trabalho é dada pela seguinte equação, também no domínio da transformada Z:

$$FiltroAntiRessonante(z) = \frac{A'z^2 + B'z^1 + C'z}{z^2} \tag{3.5}$$

Onde os parâmetros A', B' e C' se baseiam nos parâmetros do filtro ressonante original das seguintes formas:

$$A' = \frac{1}{A} \tag{3.6}$$

$$B' = \frac{-B}{A} \tag{3.7}$$

$$C' = -C \cdot A \tag{3.8}$$

Os filtros utilizados tem as seguintes funções:

a) Nasal

b) Formante 1

c) Formante 2

d) Formante 3

e) Formante 4

f) Formante 5

g) Radiação

O filtro nasal, indicado pelos blocos RNP e RNZ na figura 8 nos dão uma aproximação da nasalização das vogais e de murmúrios nasais da voz, ambos formam um par de filtros digitais ressonantes e antirressonantes.

Os filtros de formantes simulam os formantes presentes na voz e os seus parâmetros foram calculados com base nos dados contidos em Klatt (1980).

O filtro de radiação o efeito da radiação da onda sonora ao sair da boca do indivíduo falante e é basicamente um filtro digital passa-altas. Sua equação no domínio da transformada Z é dada por:

$$FiltroRadiacao = \frac{z-1}{z} \tag{3.9}$$

4 Metodologia

Este capítulo tratará dos métodos, tecnologias e técnicas que serão utilizadas no desenvolvimento do sintetizador.

4.1 Escolha das tecnologias

Para o desenvolvimento do protótipo inicial do sintetizador, foi utilizado o ambiente de desenvolvimento MATLAB (MATHWORKS, 2017), devido à sua facilidade de uso e sua ampla e confiável biblioteca de processamento de sinais e de geração de áudios a partir de sinais digitais. Como o foco do trabalho era gerar um produto de software que pudesse ser utilizado para gerar amostras para terceiros, o MATLAB foi descartado como opção de desenvolvimento completo do projeto.

Para o produto final deste trabalho entretanto, a linguagem de programação escolhida foi Python 3.6, por ser de bom conhecimento do aluno, pela sua extensa documentação e por apresentar uma imensa possibilidade de expansão devido aos diversos módulos e bibliotecas existentes. Os códigos foram desenvolvidos na IDE PyCharm Community Edition (JETBRAINS, 2017), que é de acesso gratuito.

Outra vantagem do Python que levou à sua escolha foi a facilidade montar interfaces gráficas e gerar aplicações *standalone*. Para a interface gráfica deste projeto foi utilizada a *framework* Tkinter, que está presente na maioria das instalações do interpretador do Python 3.6.

Como Python não possui suporte nativo a operações de processamento digital de sinais, foi utilizada a biblioteca externa SciPy (DEVELOPERS, 2017), uma biblioteca muito conhecida e validada para Python *open source*, voltada para operações matemáticas e científicas mas que também possui funcionalidades para geração de áudios e gráficos, combinando em apenas um pacote a maioria das funcionalidades utilizadas neste trabalho.

Para versionamento do código e para garantir a sua integridade ao longo de todo o desenvolvimento, foi utilizada a ferramenta GitHub (GITHUB, 2017). Essa escolha foi feita porque é um dos padrões atuais da indústria de desenvolvimento de software e devido à sua integração com diversas plataformas de desenvolvimento, incluindo a PyCharm (JETBRAINS, 2017) utilizada neste trabalho. Os códigos desenvolvidos estão disponíveis no Apêndice 1, seção A, e também no repositório online acessível em <https://github.com/lucas-mv/klatt>.

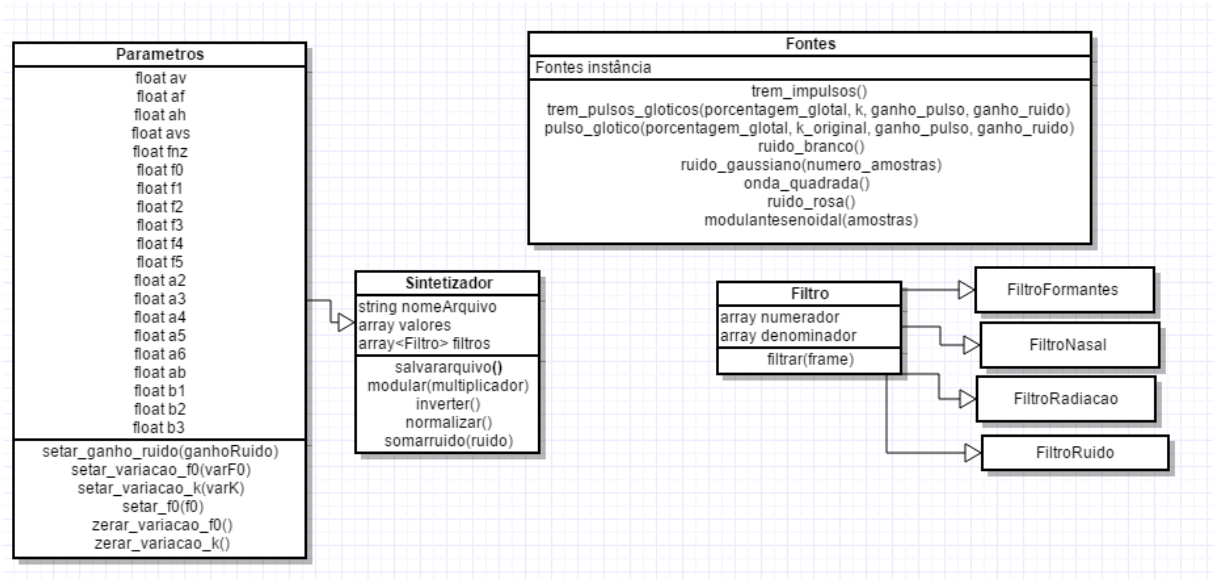

Figura 13 – UML do relacionamento das classes do sintetizador desenvolvido, criada com a ferramenta online Gliffy (GLIFFY, 2017).

4.2 Padrões de projeto

4.2.1 Padrões de linguagem

Com o objetivo de manter o código limpo, de fácil entendimento e manutenção, foi adotada a convenção de código do Python, onde as variáveis são escritas com *camel case*, as funções e métodos são separados, quando necessário, pelo caractere *underline* e as classes tem seus nomes escritos com *pascal case*. Como Python é uma linguagem fortemente baseada na identação, esta foi seguida à risca, de modo a deixar o código o mais funcional e limpo possível.

O código foi escrito em português do Brasil em sua maioria, exceto pelas chamadas de bibliotecas externas e palavras chaves da linguagem escolhida. Esta decisão foi tomada para que o código seja de fácil entendimento para futuros trabalhos nacionais e também com o intuito de servir como exemplo deste tipo de trabalho para desenvolvedores brasileiros que queiram adentrar no mundo da síntese de voz por formantes.

4.2.2 Estrutura do projeto

O projeto foi desenvolvido pelos padrões da programação orientada a objetos, seguindo os seus quatro pilares básicos: abstração, encapsulamento, herança e polimorfismo. A escolha por este padrão foi feita para facilitar a reutilização de código, leitura e manutenção.

Os objetos utilizados foram separados em classes, cada uma representando um dos aspectos do sintetizador, a figura 13 mostra como as classes relacionam entre si. Além

disso, uma importante decisão de projeto foi a de criar duas classes de uso geral: *utils* e *constantes*. A primeira contém métodos e funções de uso geral ao longo do código e a segunda contém todas as constantes usadas durante o desenvolvimento, de modo a evitar futuros problemas com variáveis definidas em tempo de execução e facilitar a leitura e compreensão do trabalho.

A estrutura do código foi pensada para que, futuramente, outros módulos possam ser acoplados a este módulo principal, como será descrito no capítulo Trabalhos Futuros 6.

5 Desenvolvimento e Resultados

Este capítulo tratará do desenvolvimento do projeto como um todo, desde a prototipagem com MATLAB até os resultados finais.

5.1 Protótipo

O desenvolvimento foi iniciado com um protótipo em MATLAB, onde foi seguido à risca o algoritmo para implementação do sintetizador de Klatt e foi comprovada a possibilidade de criação de variações deste que cumprissem os objetivos do trabalho.

Como o objetivo deste protótipo era apenas a validação da teoria e da metodologia, os seus resultados foram pouco satisfatórios, gerando vozes que não se assemelham muito com a realidade, porém serviram como base para os desenvolvimentos futuros.

Após a validação da teoria, o desenvolvimento do trabalho em si foi iniciado, levando em consideração a metodologia descrita na seção 4.2.

5.2 Desenvolvimento do projeto

Muitas imagens a partir do *software* desenvolvido estão presentes neste capítulo. Para facilitar o entendimento do leitor, a tabela 1 contém os valores dos parâmetros utilizados na geração destes resultados.

5.2.1 Criação do modelo do pulso

O modelo para criação de pulsos glóticos descrito em Klatt (1980) foi implementado com sucesso e os resultados se aproximaram dos apresentados na figura 9, porém esta abordagem de geração dos pulsos a partir de filtros trouxe diversas complicações. A inserção de ruídos não estava sendo precisa o suficiente, já que esta depende da modulação em amplitude do ruído com a modulante sendo uma onda quadrada de frequência igual F0 (ver figura 8 para maiores detalhes), ou seja, a inserção do ruído é baseada no ciclo de trabalho desta onda modulante e não no pulso em si. Outro problema que tivemos foi a falta de possibilidades de variações da forma de onda obtida com esta abordagem. O controle do período de fechamento do pulso glótico é complexo e este método basicamente apenas permite a seleção da frequência fundamental F0 de modo simples. A seção 3.2 contém uma descrição mais profunda deste método e das suas imperfeições identificadas.

Com isso, a abordagem foi modificada para gerar os pulsos segundo Fant (1979). Os resultados obtidos foram mais próximos de pulsos reais e podem ser analisados na figura 14. Como citado anteriormente, desta forma temos controle maior dos tempos de

Tabela 1 – Parâmetros de entrada do *software* utilizados na geração das imagens.

Parâmetro	Valor	Unidade
F0	150	Hz
Tempo de simulação	2	s
Período de amostragem	0.0001	s
Variação de F0	5	%
Variação de k	50	%
Ganho do ruído	0.5	Adimensional
Ganho do pulso	1.0	Adimensional
k	1.0	Adimensional
AN	0.0	dB
A1	0.0	dB
F1	620	Hz
F1	80	Hz
F2	1220	Hz
B2	50	Hz
F3	2550	Hz
B3	140	Hz
B4	250	Hz
B5	200	Hz
FNP	250	Hz
BNP	250	Hz
BNZ	100	Hz

Fonte: Dados retirados de Klatt (1980) e gerados empíricamente pelos autores.

Nota: Os demais parâmetros não citados aqui podem ser facilmente encontrados em Klatt (1980, p. 976).

silêncio e conseguimos configurar corretamente quando começam e terminam, possibilitando a inclusão de ruídos após o término do pulso, o que também se assemelha mais ao comportamento real.

Além disso, foi adicionado um controle de intensidade do pulso, podendo reduzir ou aumentar o seu valor final caso necessário. Para as amostras geradas neste trabalho e as inclusas na seção B, foi utilizado o controle de pulsos em sua maior intensidade.

5.2.2 Modelo e inserção do ruído

Neste trabalho, o modelo do ruído utilizado tem distribuição gaussiana, como indicado em Klatt (1980). Para gerar esta distribuição, foi utilizada o módulo Numpy da biblioteca SciPy (DEVELOPERS, 2017), que possui além deste, diversos outros modelos

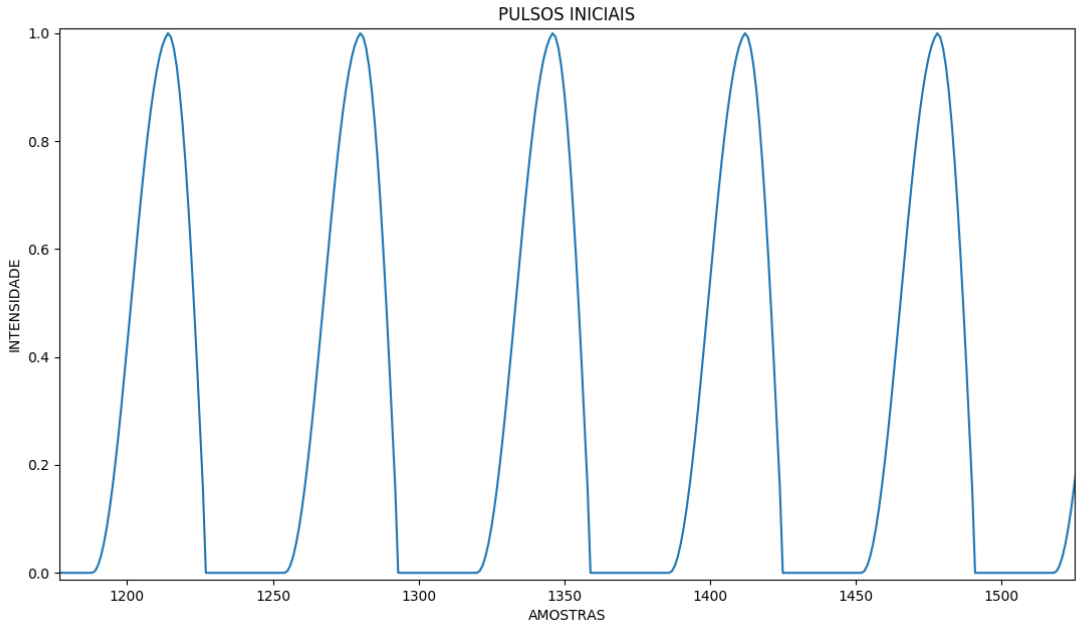

Figura 14 – Pulsos gerados com o algoritmo descrito em Fant (1979). Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

de ruído, facilitando a sua troca caso este seja um parâmetro controlável futuramente.

Após o término de um pulso, as amostras de ruído são geradas e somadas aos valores iniciais durante o período de maior aproximação das pregas vocais, como pode ser visto na figura 15. Este comportamento simula o real, já que, se neste período não houver o fechamento total das pregas vocais, haverá uma fenda que gerará ruído de fricção.

Foi adicionado um parâmetro de controle de intensidade do ruído, similar ao controle de intensidade dos pulsos, para possibilitar a geração de vozes pouco ou nada ruidosas, uma das premissas originais deste trabalho. Para gerar as figuras contidas aqui e as amostras da seção B, este controle foi configurado como 50% do valor máximo total. Devido à natureza do ruído gaussiano e de modo a garantir que este não ultrapasse o valor máximo dos pulsos e fique sempre contido neste intervalo, os valores gerados são normalizados entre 0% e o valor de controle.

5.2.3 Controle de variação de F0 e de k

Para melhorar a qualidade da voz sintetizada e também dar mais controle para o usuário final, foi gerada uma rotina de variação de F0 e de k pulso a pulso. Desta forma cada pulso será diferente do outro não somente no ruído como também em pequenas variações de frequência fundamental e de velocidade de queda. A figura 16 mostra um pulso com estes controles de variação habilitados.

Para gerar as amostras contidas na seção B e da figura 16 os parâmetros foram

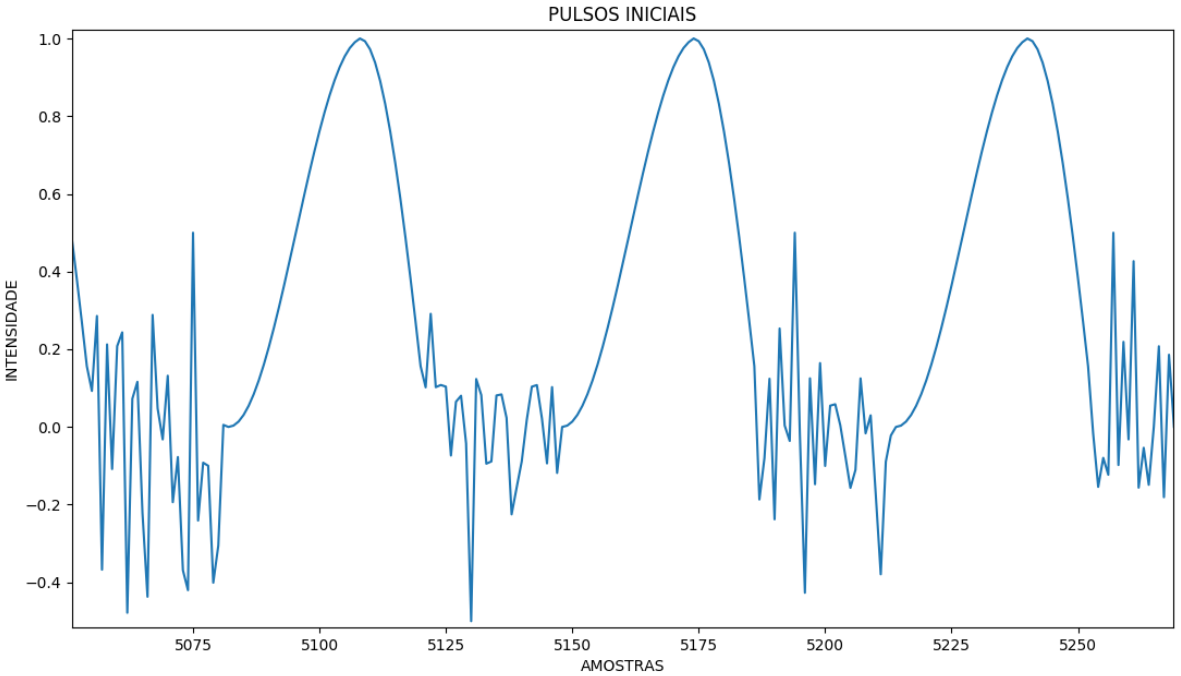

Figura 15 – Pulsos gerados com adição dos ruídos. Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

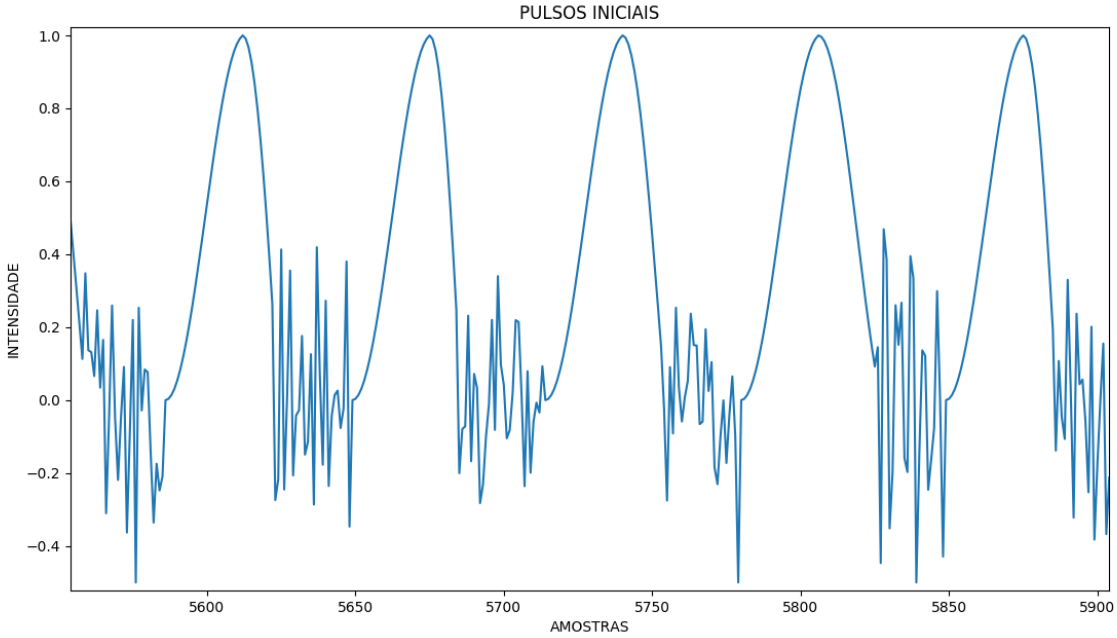

Figura 16 – Pulsos com ruído adicionado e com variações sutis na frequência fundamental (F0) e na velociadade de descida (k). Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

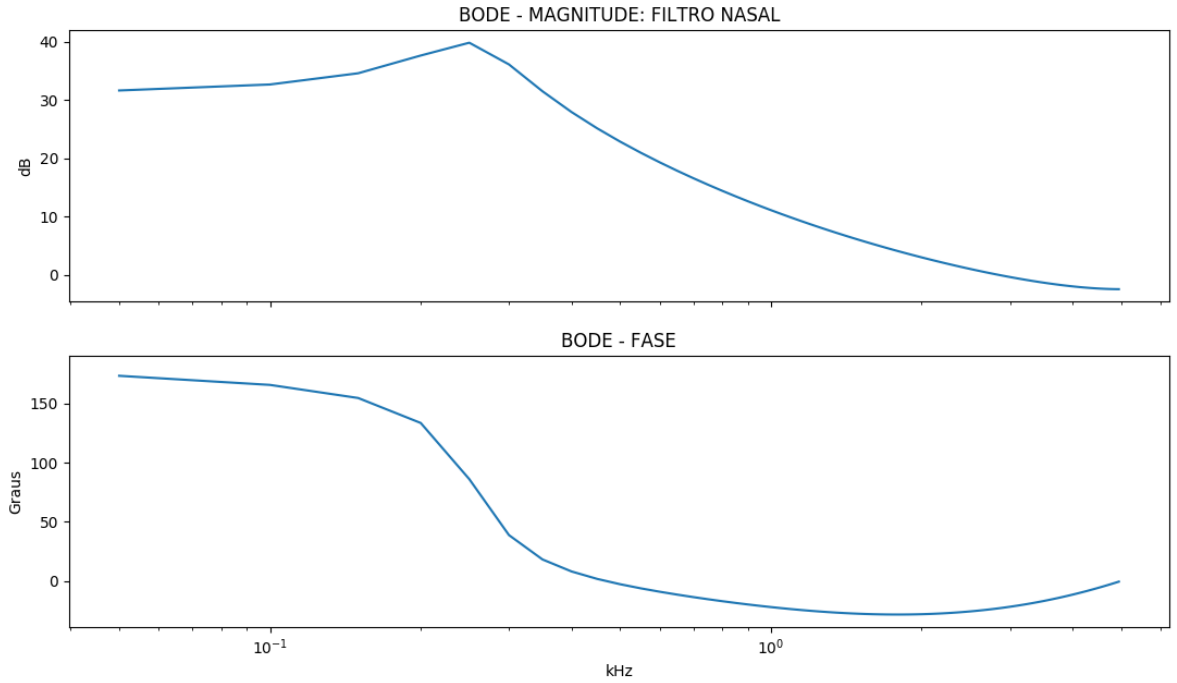

Figura 17 – Diagrama de Bode do filtro que simula o comportamento nasal desenvolvido. Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

configurados da seguinte forma:

$$VariaçãoPercentualF0 = 5\% \tag{5.1}$$

$$VariaçãoPercentualK = 50\% \tag{5.2}$$

Estes valores foram adequados ao longo de vários testes e apresentam uma variação factível destes parâmetros e que dão uma melhor sonoridade ao áudio gerado.

5.2.4 Arquitetura dos filtros e filtragem dos sinais gerados

Para a implementação dos filtros, foi usada a estrutura de herança descrita em detalhes na seção 4.2 e o módulo *signal* da biblioteca SciPy (DEVELOPERS, 2017), bem como a teoria descrita em 3.3. Os parâmetros de cada um dos filtro podem ser facilmente modificados no sintetizador desenvolvido, alterando os dados de entrada da classe *Parametros*, cuja estrutura também é explicada com mais detalhes na seção 4.2.

As figuras 17, 18 e 19 mostram os diagramas de Bode dos filtros gerados. É importante salientar que os resultados da saída dos filtros serão normalizados para valores entre 0.0 e 1.0, fazendo com que o ganho destes filtros seja apenas uma forma de selecionar as frequências de interesse.

Seguindo o fluxo de operação do sintetizador, os pulsos serão gerados, adicionados de ruído e, em seguida, filtrados por estes em cascata, um após o outro.

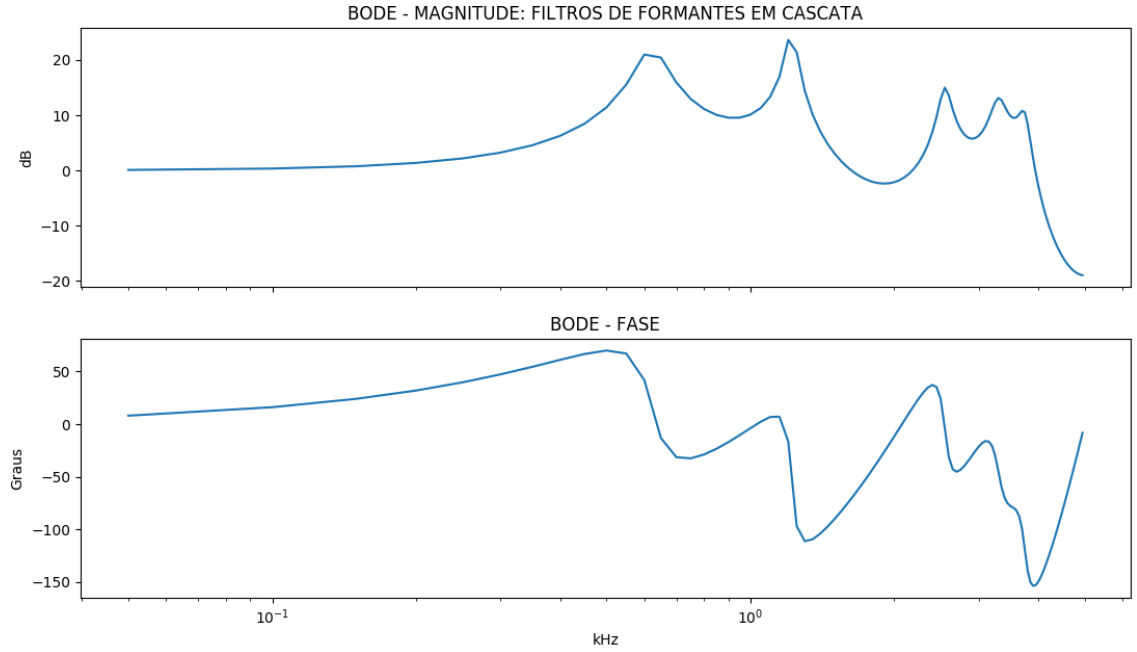

Figura 18 – Diagrama de Bode do filtro de formantes desenvolvido, onde os formantes 1, 2, 3, 4 e 5 já estão em cascata. Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

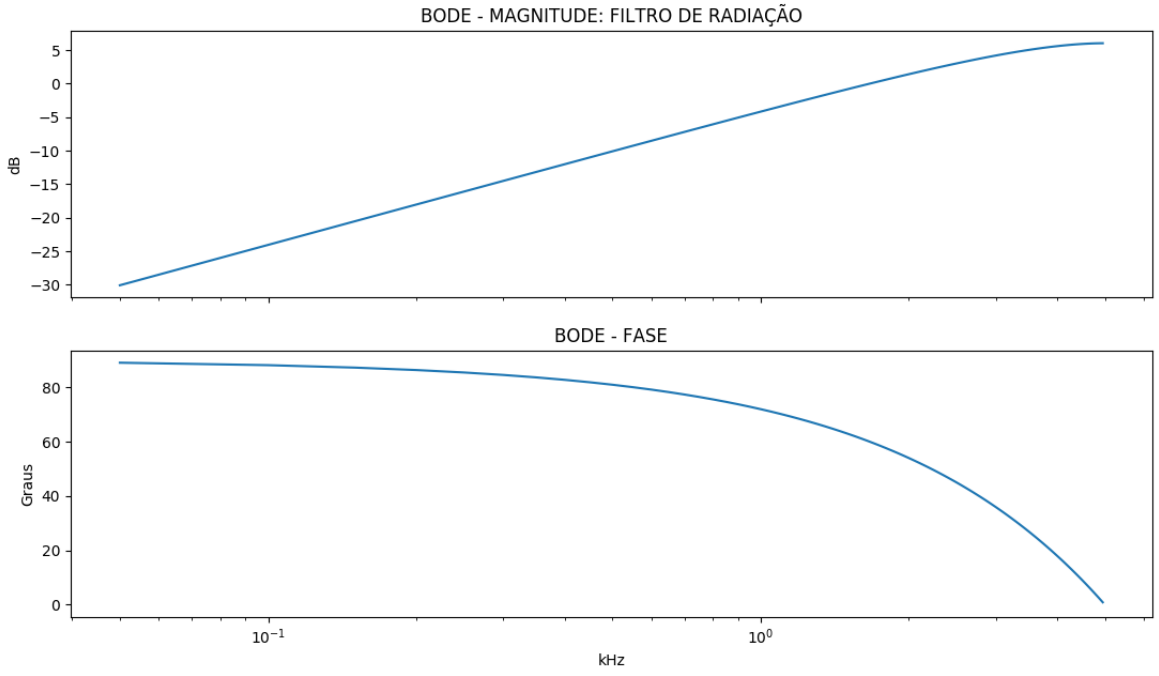

Figura 19 – Diagrama de Bode do filtro de radiação desenvolvido. Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

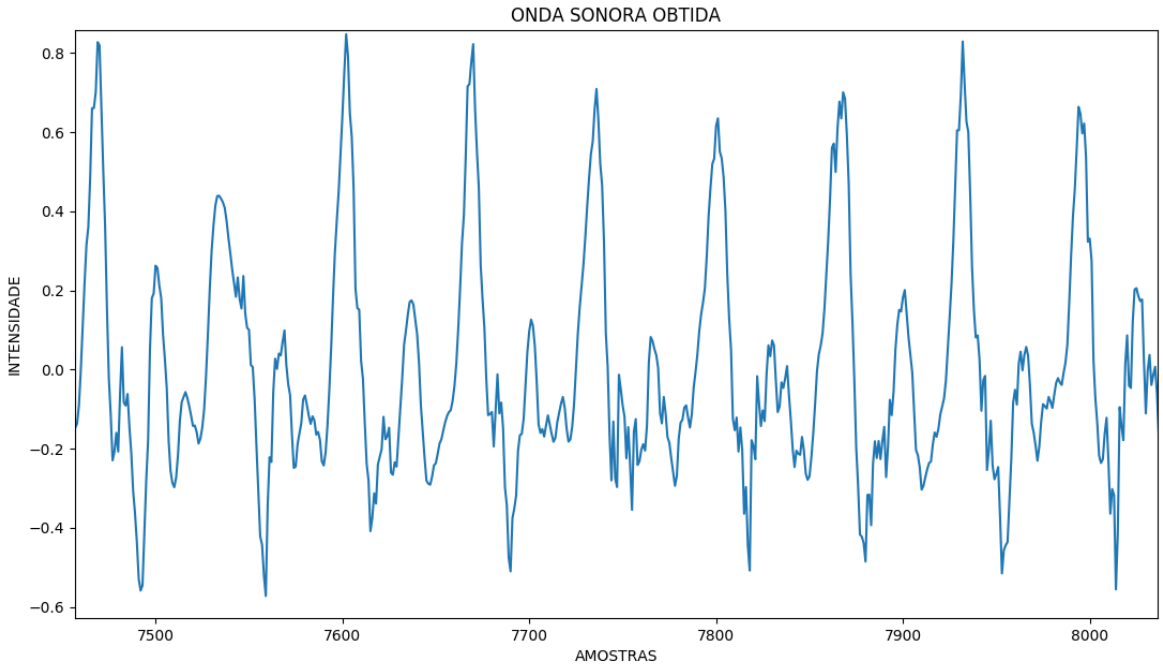

Figura 20 – Parte da onda sonora final gerada pelo sintetizador. Imagem gerada com o módulo Matplotlib, da biblioteca SciPy (DEVELOPERS, 2017).

5.2.5 Geração do arquivo de áudio

Após todos esses processos o arquivo de áudio é gerado e salvo com o nome dado pelo usuário no começo da execução do programa. Para gerar este arquivo foi utilizado o módulo *wave* da biblioteca SciPy (DEVELOPERS, 2017). A figura 20 mostra uma forma de onda gerada por este sintetizador.

5.2.6 Interface gráfica

Foi desenvolvida uma interface gráfica básica para usuários que desejam alterar alguns parâmetros do programa antes de sintetizar, que se mostrou bem útil para testar novas configurações e validar resultados. Foi desenvolvida com a *framework* Tkinter, presente na maioria das versões do Python 3.6.

Os parâmetros que o usuário pode modificar com esta interface são:

a) Nome do arquivo

b) Tempo de simulação

c) Variação de F0

d) Variação de K

e) Ganho do ruído

f) Ganho do pulso

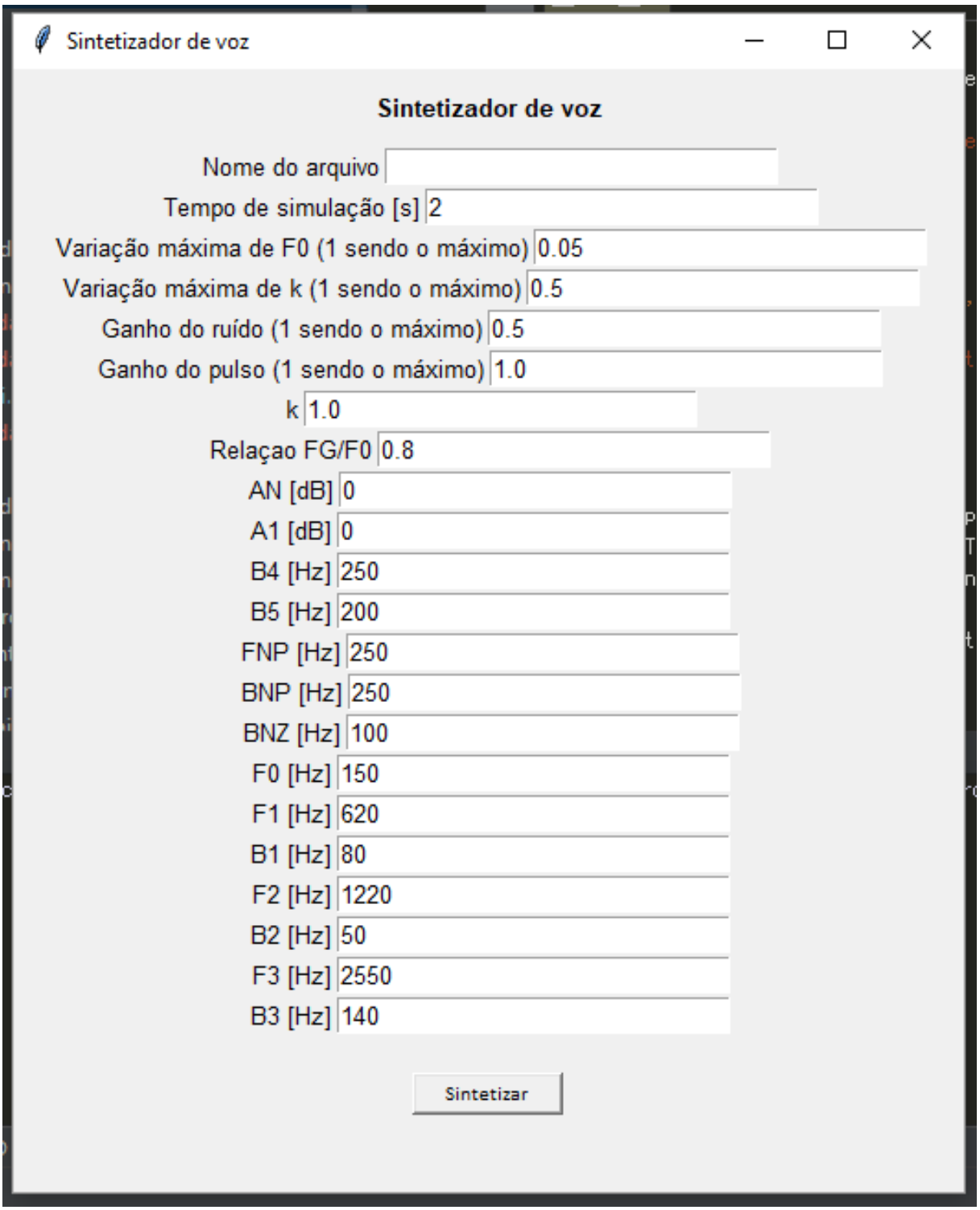

Figura 21 – Interface gráfica básica desenvolvida para o sistema, permite que o usuário altere alguns parâmetros antes de sintetizar a voz.

g) Relação FG/F0

h) K

i) AN

j) A1

k) B4

l) B5

m) FNP

n) BNP

o) BNZ

p) F0

q) F1

r) B1

s) F2

t) B2

u) F3

v) B3

Todos estes são parâmetros do sintetizador de Klatt, que estão explicados em sua totalizade em Klatt (1980) e Fant (1979), exceto pela "Relação FG/F0", que representa a porcentagem de F0 relativa à frequência do pulso glótico FG.

Ao clicar em no botão "Sintetizar" o *software* tentará sintetizar a voz com os parâmetros inseridos e, ao terminar exibirá a mensagem "Áudio sintetizado!" abaixo do botão e salvará o arquivo na mesma pasta em que está o executável. Caso ocorra algum erro a mensagem exibida será "Erro:", seguida da descrição do erro.

6 Trabalhos futuros

Este capítulo tratará das melhorias futuras que podem ser feitas, com relação a experiência do usuário, inclusão de funcionalidades e geração de conhecimento com este trabalho.

6.1 Experiência do usuário

Vários módulos podem ser adicionados a este sintetizador para melhorar a experiência do usuário final, sendo os principais os seguintes:

a) Melhoria da interface gráfica da aplicação *standalone*

b) Criação de um sistema web e API para acessar o sintetizador

c) Criação de um aplicativo *mobile* que execute as rotinas de sintetização

A seguir temos uma curta explicação de cada um destes itens e de como poderiam ser feitos.

6.1.1 Melhoria da interface gráfica

A *framework* Tkinter permite a criação de interfaces gráficas muito mais complexas e responsivas do que a atual. Nada impediria de futuramente a GUI ser migrada para outra *framework* que permita o desenvolvimento de um design melhor.

Melhorias possíveis de serem feitas e de complexidade não muito elevada seriam a melhor formatação da tela e o uso de entradas de dados mais amigáveis ao usuário, de modo a impedir que este cometa erros de entrada de parâmetros.

A seleção de caminho onde salvar o arquivo gerado e a inclusão de um menu de ajuda que explique cada um dos parâmetros também são melhorias muito interessantes para evitar frustrações ao operar o *software*.

6.1.2 Sistema web e API

A criação de uma API que encapsula o módulo do sintetizador não seria muito complicado, existem muitas bibliotecas em Python voltadas para esta funcionalidade, a mais famosa atualmente sendo a *framework* Django, acessível em <https://www.djangoproject.com>. Este pacote possui uma boa integração com a IDE PyCharm (JETBRAINS, 2017) utilizada para o desenvolvimento do projeto, facilitando ainda mais esta melhoria.

Esta API poderia ser acessada de diversas formas, um sistema web seria uma ótima opção já que os usuários teriam sempre uma versão atualizada do sistema ao acessar o

sítio. Outra vantagem do sistema web são as inúmeras bibliotecas de desenvolvimento e de componentes visuais, bem como a facilidade de desenvolver designs amigáveis.

6.1.3 Aplicativo

Atualmente o desenvolvimento de aplicativos se tornou um processo bem simples e este poderia ser feito de diversas formas. Uma possibilidade seria a criação de uma API, assim como no item anterior, e criar um aplicativo que acesse esta. Outra possibilidade seria encapsular o módulo do sintetizador e gerar um aplicativo *standalone* diretamente com este, utilizando alguma *framework* de desenvolvimento *mobile* própria do Python, como a Kivy, acessível em <https://kivy.org>.

Caso a abordagem da API seja escolhida, temos a possibilidade de desenvolvimento nativo para cada plataforma ou o uso de ferramentas multiplataformas, como Kony ou Xamarin, respectivamente acessíveis nos sítios <http://www.kony.com/pt> e <https://www.xamarin.com>.

6.2 Funcionalidades futuras

Importantes funcionalidades que podem ser implementadas futuramente são a geração de sonogramas das vozes sintetizadas e medições destas, principalmente da relação sinal-ruído (*SNR*), da variação aleatória ciclo-a-ciclo da frequência fundamental F0 e da amplitude dos pulsos glóticos, respectivamente conhecidos como *jitter* e *shimmer*.

O sonograma poderia ser facilmente gerado com o módulo matplotlib da biblioteca SciPy (DEVELOPERS, 2017), que já é usado neste projeto. As mudanças necessárias para incluir esta funcionalidade seriam pequenas e seria uma grande melhoria para usuários finais que usam esta ferramenta para avaliar a voz. A medição de SNR também pode ser feita com a biblioteca SciPy (DEVELOPERS, 2017), usando módulo stats.

Outra abordagem seria, a partir do sonograma fazer as medições, seguindo o método descrito em Vieira, Sansao e Yehia (2014), que obteve resultados robustos, com diversos cenários diferentes, além de ter a vantagem de ser um método específico de medições em vozes disfônicas.

6.3 Geração de conhecimento

A síntese de vozes disfônicas é uma área em crescimento e este trabalho pode ser desenvolvido de diversas formas para acrescentar a esta.

Uma das formas que este trabalho pode contribuir com a área é fazendo testes de percepção com as diversas amostras geradas, contidas na seção B. Estes testes nos

mostrariam os limites da metodologia desenvolvida para gerar vozes similares às reais e dariam informações sobre onde é possível melhorar o algoritmo utilizado.

Os testes de percepção alinhados com as medições descritas na seção 6.2 também forneceriam informações sobre os ganhos de qualidade e de customização da voz obtidos ao utilizar o método de gerar pulsos descrito em Fant (1979) e os parâmetros de controle de ruído, intensidade dos pulsos, variação de k, F0 e da relação FG/F0.

Ainda sobre os testes de percepção, estes podem ser utilizados para avaliarmos a menor diferença perceptível com relação aos parâmetros envolvidos, trazendo assim melhor entendimento da relação destes com a qualidade da voz sintetizada. Tais estudos seriam importantes para que os usuários profissionais de fonoaudiologia tenham, nesta ferramenta, auxílio para quantificação de perturbações na voz associadas a alterações nas vibrações das pregas vocais.

Existe bastante possibilidade de melhoria do *software* com relação à escolha dos parâmetros iniciais para a geração das vozes disfônicas, e é da vontade do desenvolvedor continuar este trabalho de modo a fazer uma compilação de parâmetros ideais para síntese de cada tipo de patologia da voz, porém isso só seria possível após os teste de percepção citados acima.

Outro estudo futuro que pode ser feito a partir deste trabalho é validar analiticamente, por meio das medições indicadas na seção 6.2, a existência dos fenômenos de *jitter* e *shimmer* nas vozes sintetizadas utilizando os controles de variação de F0 e k pulso a pulso. Tais fenômenos foram observados empiricamente durante o desenvolvimento do projeto, mas não houve tempo para realizar as medições necessárias e comprovar que a inserção destes controles foram as reais causas destes.

7 Conclusão

Este capítulo tratará das conclusões a serem tiradas deste trabalho, bem como das discussões que ocorreram ao longo do seu desenvolvimento.

Várias conclusões podem ser tiradas sobre as diversos áreas deste trabalho, abaixo segue uma lista dos principais tópicos a serem abordados nessa seção:

a) Desenvolvimento das habilidades profissionais e uso dos conhecimentos adquiridos no curso;

b) Qualidade do código;

c) Qualidade da voz sintetizada;

d) Melhorias observadas com o uso dos controles desenvolvidos;

e) Uso da metodologia descrita em Fant (1979).

7.1 Desenvolvimento das habilidades profissionais

Este trabalho sintetizou diversos conhecimentos adquiridos durante o curso, desde a parte de estruturação e organização do projeto até as habilidades de escrita de relatórios técnicos.

No início do desenvolvimento foi feito um estudo para chegar à estrutura descrita na seção 4.2, que só foi possível devido aos conhecimentos de estruturação de projetos em geral e mais especificamente de programação orientada a objetos, conhecimentos estes que foram adquiridos durante diversas matérias do curso de Engenharia Elétrica.

Muitas das técnicas de programação utilizadas neste projeto também foram aprendidas durante o curso, em especial nas disciplinas “Programação Orientada a Objetos” e “Algoritmos e Estruturas de Dados II”, porém muitas foram adquiridas com a prática deste trabalho, principalmente as relacionadas à linguagem Python.

O uso de filtros digitais também foi essencial para o desenvolvimento do projeto, cuja compreensão só foi possível graças às disciplinas “Processamento de Sinais”, “Análise de Circuitos Elétricos II”, “Análise de Circuitos Elétricos III” e “Análise de Sistemas Lineares”.

As habilidades de escrita de trabalhos acadêmicos, que foram muito utilizadas neste texto, também foram estudadas no curso, extensamente nas diversas disciplinas de laboratório, mas também nos projetos de pesquisa que foram desenvolvidos durante o tempo como graduando da UFMG.

7.2 Qualidade do código

Qualidade do código foi uma preocupação ao longo da evolução do projeto, visando manter a sua manutenção e compreensão sempre fácil tanto para o desenvolvedor quanto para terceiros que pudessem vir a analisá-lo posteriormente.

É da compreensão do autor que o código desenvolvido apresenta boa qualidade e respeita boas práticas da programação orientada a objetos e da linguagem escolhida, apesar de que sempre existem espaços para melhoria.

Houveram algumas mudanças na estrutura do código durante o desenvolvimento, porém nunca foram quebras explícitas da arquitetura do *software* que dificultariam a manutenção do código.

7.3 Qualidade da voz sintetizada

O sintetizador desenvolvido conseguiu sintetizar corretamente vozes disfônicas com ruídos, os resultados foram bastante similares aos observados em pacientes com algumas patologias da voz.

Como grande parte do tempo gasto foi dedicado à implementação da teoria e da metodologia, ainda existe muita possibilidade de melhoria, principalmente na seleção dos parâmetros ideais para cada patologia da voz, como discutido na seção 6.3, e para cada perfil de pessoa falante.

7.4 Controles adicionais desenvolvidos

Com adição aos parâmetros já definidos em Klatt (1980) e em Fant (1979), foram desenvolvidos os controles adicionais de intensidade de ruído e do pulso, variação de F0 e de k e da relação FG/F0. Tais parâmetros foram inseridos para termos melhor controle sobre a voz a ser gerada.

Os resultados obtidos usando estes controles foram melhores do que os esperados, principalmente com relação à variação de F0 e de k. A variação de F0 pulso a pulso levou à geração de *jitter* no áudio sintetizado, enquanto a variação de k gerou *shimmer*, dois fenômenos presentes em vozes reais e que incorporaram verossimilhança ao produto final. Seria necessário fazer um estudo de medições destes fenômenos nas vozes sintetizadas para comprovar analiticamente a presença destes dois fenômenos, como descrito na seção 6.3.

O controle de intensidade do ruído se provou essencial para modular a voz de modo a simular a intensidade da patologia a ser sintetizada, enquanto o controle da relação FG/F0 se mostrou a peça chave para controlar a assimetria do pulso glótico.

7.5 Uso da metodologia descrita em Fant (1979)

Este ponto foi muito discutido nas seções anteriores, mas é de grande importância para o projeto. Este método permitiu administrar os pulsos glóticos gerados de tal forma que a maioria dos controles citados anteriormente não seria possível sem ele.

O fato de conseguir controlar onde a duração da fase fechada do ciclo glótico se mostrou essencial para a inserção correta do ruído, ao mesmo tempo que a gestão da velocidade de descida do pulso, relacionado à variável k, adicionou muitas possibilidades de customização da voz a ser sintetizada.

O método descrito em Klatt (1980) é extremamente eficiente para gerar pulsos e o fato de utilizar filtros digitais é um grande adicional, já que estes podem ser inseridos em cascata com os filtros seguintes, o que faz a implementação deste método ser bastante simples. Apesar disso, para a síntese de vozes disfônicas ele se provou pouco eficiente. Suas possibilidades de controle do pulso são mínimas e requerem pós processamento dos sinais gerados, o que aumenta muito o custo computacional e a complexidade do problema.

Em suma, o uso dessa metodologia não só facilitou algumas partes do desenvolvimento do projeto, como também adicionou muitas possibilidades de expansão ao mesmo.

7.6 Considerações finais

O trabalho desenvolvido apresentou resultados satisfatórios e grandes possibilidades de evoluções futuras, que são de grande interesse do aluno.

Vários conhecimentos adquiridos ao longo do curso de Engenharia Elétrica foram sintetizados neste projeto e várias habilidades profissionais, que não serão descartadas de forma alguma, foram assimiladas pelo aluno, conhecimentos esses que não seriam obtidos de nenhuma outra forma.

É com grandes esperanças de que este texto e os códigos desenvolvidos aqui contidos sejam utilizados futuramente e que sirvam para gerar conhecimentos, seja por pesquisadores ou apenas por pessoas com amor à área, que este trabalho é encerrado.

Referências

DEVELOPERS, S. Biblioteca open souce para Python, *SciPy.org*. 2017. Disponível em: <https://www.scipy.org/index.html>. Acesso em: 1 jul. 2017. Citado 10 vezes nas páginas 11, 12, 35, 40, 41, 42, 43, 44, 45 e 50.

FANT, G. Vocal source analysis-a progress report. *STL-QPSR*, v. 20, n. 3-4, p. 31–53, 1979. Citado 11 vezes nas páginas 11, 18, 21, 30, 39, 41, 47, 51, 53, 54 e 55.

FANT, G.; MARTONY, J. Instrumentation for parametric synthesis (ove-ii). *KTH Speech Transmission Laboratory: Quarterly Progress and Status Report, 2*, v. 3, p. 18–19, 1962. Citado 4 vezes nas páginas 11, 24, 25 e 30.

GITHUB. Gerenciador de repositórios de código GitHub, *GitHub*. 2017. Disponível em: <https://github.com>. Acesso em: 1 jul. 2017. Citado na página 35.

GLIFFY. Ferramenta online para desenvolvimento de diagramas, *Gliffy*. 2017. Disponível em: <https://www.gliffy.com>. Acesso em: 1 jul. 2017. Citado 2 vezes nas páginas 11 e 36.

GOLD, B.; RABINER, L. Analysis of digital and analog formant synthesizers. *IEEE Transactions on Audio and Electroacoustics*, IEEE, v. 16, n. 1, p. 81–94, 1968. Citado na página 26.

HUNT, A. J.; BLACK, A. W. Unit selection in a concatenative speech synthesis system using a large speech database. In: IEEE. *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*. [S.l.], 1996. v. 1, p. 373–376. Citado na página 26.

JETBRAINS. Página da IDE para desenvolvimento de Python Pycharm, *PyCharm*. 2017. Disponível em: <https://www.jetbrains.com/pycharm/>. Acesso em: 1 jul. 2017. Citado 2 vezes nas páginas 35 e 49.

KLATT, D. H. Software for a cascade/parallel formant synthesizer. *the Journal of the Acoustical Society of America*, ASA, v. 67, n. 3, p. 971–995, 1980. Citado 16 vezes nas páginas 11, 17, 21, 23, 26, 27, 29, 30, 31, 32, 33, 39, 40, 47, 54 e 55.

LILJENCRANTS, J. The ove iii speech synthesizer. *IEEE Transactions on Audio and Electroacoustics*, IEEE, v. 16, n. 1, p. 137–140, 1968. Citado 3 vezes nas páginas 11, 25 e 30.

LUCERO, J. C. et al. A vocal fold model for disordered voice synthesis. In: *9th International Conference in Voice Physiology and Biomechanics*. Salt Lake City: CNPq and F.R.S./F.N.R.S., 2014. (ICVPB). Disponível em: <http://www.cic.unb.br/~lucero/disorder.html>. Acesso em: 1 jul. 2017. Citado 2 vezes nas páginas 21 e 26.

MATHWORKS. Página do ambiente de computação matemática MATLAB, *MATLAB*. 2017. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 1 jul. 2017. Citado na página 35.

STYLIANOU, Y.; SYRDAL, A. K. Perceptual and objective detection of discontinuities in concatenative speech synthesis. In: IEEE. *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*. [S.l.], 2001. v. 2, p. 837–840. Citado na página 26.

TARNÓCZY, T. The speaking machine of wolfgang von kempelen. *The Journal of the Acoustical Society of America*, Acoustical Society of America, v. 21, n. 4, p. 461–461, 1949. Citado na página 23.

TRAUNMÜLLER, H. History of speech synthesis, 1770-1970. *Web page http://www. ling. su. se/staff/hartmut/kemplne. htm. Last checked*, p. 05–11, 2005. Citado 4 vezes nas páginas 11, 23, 24 e 25.

VIEIRA, M. N. Nota de aula, *Sobre a Anatomia, Fisiologia e Patologia da Laringe*. Departamento de Física - ICEx - UFMG: [s.n.], 1999. Citado 3 vezes nas páginas 11, 27 e 28.

VIEIRA, M. N. Princípios da Produção e Análise de Voz. In: *Princípios da Produção e Análise de Voz*. Departamento de Física/ICEx/UFMG: Departamento de Física/ICEx/UFMG, 2004. p. 1–11. Citado 2 vezes nas páginas 11 e 28.

VIEIRA, M. N.; SANSAO, J. P. H.; YEHIA, H. C. Measurement of signal-to-noise ratio in dysphonic voices by image processing of spectrograms. *Speech Communication*, Elsevier, v. 61, p. 17–32, 2014. Citado 2 vezes nas páginas 26 e 50.

Apêndices

APÊNDICE A – Códigos

Todos os códigos contidos aqui também podem ser encontrados no repositório online acessível em: <https://github.com/lucas-mv/klatt>, acesso em 01 de jul. de 2017.

A.1 Códigos do sintetizador

A.1.1 Main

```
"""
Modulo para testes do sintetizador
"""

from synth import utils
import synth.sintetizador as sintetizador
import  synth.parametros as params
import synth.constantes as ctes

def main():
    som = sintetizador.sintetizar('audios/f0=150-modSin-noise', 'a')
    utils.plotar_formaonda(som)
    utils.mostrar_plots()

    # params.zerar_variacao_f0()
    # params.zerar_variacao_k()
    # for f0 in range(150, 220, 10):
    #     for ganhoRuido in range(0,100,5):
    #         params.setar_ganho_ruido(ganhoRuido/100)
    #         params.setar_f0(f0)
    #         filename = 'audios/variacaoApenasRuido/f0='+ str(ctes.ParametrosConstantes.F0) +'-varNoiseOnly-noiseGain='\
    #                     + str(ctes.Gerais.GANHO_RUIDO)
    #         sintetizador.sintetizar(filename, 'a')
    #
    # params.setar_variacao_f0(0.05)
    # for f0 in range(150, 220, 10):
    #     for ganhoRuido in range(0, 100, 5):
    #         params.setar_ganho_ruido(ganhoRuido / 100)
    #         params.setar_f0(f0)
    #         filename = 'audios/variacaoF0/f0=' + str(ctes.ParametrosConstantes.F0) + '-varf0-noiseGain=' \
    #                     + str(ctes.Gerais.GANHO_RUIDO) + '-varF0=' + str(ctes.Gerais.VARIACAO_F0)
```

```
        #         sintetizador.sintetizar(filename, 'a')
33      #
        # params.zerar_variacao_f0()
        # params.setar_variacao_k(0.5)
        # for f0 in range(150, 220, 10):
37      #     for ganhoRuido in range(0, 100, 5):
        #         params.setar_ganho_ruido(ganhoRuido / 100)
        #         params.setar_f0(f0)
        #         filename = 'audios/variacaoK/f0=' + str(ctes.
    ParametrosConstantes.F0) + '-varK-noiseGain=' \
41      #                    + str(ctes.Gerais.GANHO_RUIDO) + '-varK=' + str
    (ctes.Gerais.VARIACAO_K)
        #         sintetizador.sintetizar(filename, 'a')

    if __name__ == "__main__":
45      main()
```

synth/main.py

A.1.2 Filtros

```
    """
    Conjunto de codigos que definem o funcionamento dos filtros ressonantes
       do sintetizador.
 3  """

    import synth.parametros as params
    import synth.constantes as ctes
 7  import math
    import numpy as np
    from scipy import signal


11
    class Filtro:

        _numerador = None
15      _denominador = None

        @property
        def numerador(self):
19          return self._numerador

        @property
        def denominador(self):
23          return self._denominador

        def filtrar(self, frame):
```

```
            funcao_transferencia = signal.TransferFunction(self._numerador,
        self._denominador, dt=ctes.Amostragem.TEMPO_AMOSTRAGEM)
            x, y = signal.dlsim(funcao_transferencia, frame)
            return y


class FiltroFormantes(Filtro):

    def __init__(self, vogal):
        Filtro.__init__(self)
        parametros = params.ParametrosCascata(vogal)
        self._numerador, self._denominador = bloco_formantes(parametros)


class FiltroFontes(Filtro):

    def __init__(self, vogal, av, avs):
        Filtro.__init__(self)
        parametros = params.ParametrosCascata(vogal, av, avs)
        self._numerador, self._denominador = bloco_fonte(parametros)


class FiltroNasal(Filtro):

    def __init__(self, vogal):
        Filtro.__init__(self)
        parametros = params.ParametrosCascata(vogal)
        self._numerador, self._denominador = bloco_nasal(parametros)


class FiltroRadiacao(Filtro):

    def __init__(self):
        Filtro.__init__(self)
        self._numerador, self._denominador = bloco_radiacao()


class FiltroRuido(Filtro):

    def __init__(self):
        Filtro.__init__(self)
        self._numerador, self._denominador = bloco_ruido()


def calcularabc(bw, f):
    t = ctes.Amostragem.TEMPO_AMOSTRAGEM
    c = -1.0 * math.exp(-2.0 * np.pi * bw * t)
```

```
        b = 2.0 * math.exp(-1.0 * np.pi * bw * t) * math.cos(2.0 * np.pi * t
        * f)
        a = 1.0 - b - c
        return a, b, c
75

    def montar_numden(bw, f):
        a, b, c = calcularabc(bw, f)
79      num = [a, 0.0, 0.0]
        den = [1.0, -1.0*b, -1.0*c]
        return num, den

83
    def montar_numden_antiressonante(bw, f):
        a, b, c = calcularabc(bw, f)
        a_anti = 1.0/a
87      b_anti = -1.0*b/a
        c_anti = -1.0*c*a
        num = [a_anti, b_anti, c_anti]
        den = [1.0, 0.0, 0.0]
91      return num, den


    def bloco_fonte(parametros):
95      num_rgp, den_rgp = montar_numden(ctes.ParametrosConstantes.BGP, ctes
       .ParametrosConstantes.FGP)
        num_rgs, den_rgs = montar_numden(ctes.ParametrosConstantes.BGS, ctes
       .ParametrosConstantes.FGS)
        num_rgz, den_rgz = montar_numden_antiressonante(ctes.
       ParametrosConstantes.BGZ, ctes.ParametrosConstantes.FGZ)
        for indice in range(len(num_rgs)):
99          num_rgs[indice] = num_rgs[indice] * parametros.av
        for indice in range(len(num_rgz)):
            num_rgz[indice] = num_rgz[indice] * parametros.avs
        num_paralelo = [num_rgs[0] + num_rgz[0], num_rgz[1], num_rgz[2]]
103     den_paralelo = [den_rgz[0] + den_rgs[0], den_rgs[1], den_rgs[2]]
        num_fonte = np.convolve(num_rgp, num_paralelo)
        den_fonte = np.convolve(den_rgp, den_paralelo)
        return num_fonte, den_fonte
107

    def bloco_nasal(parametros):
        num_rnp, den_rnp = montar_numden(ctes.ParametrosConstantes.BNP, ctes
       .ParametrosConstantes.FNP)
111     num_rnz, den_rnz = montar_numden_antiressonante(ctes.
       ParametrosConstantes.BNZ, parametros.fnz)
        num_nasal = np.convolve(num_rnp, num_rnz)
```

```
        den_nasal = np.convolve(den_rnp, den_rnz)
        return num_nasal, den_nasal


    def bloco_formantes(parametros):
        num_1, den_1 = montar_numden(parametros.b1, parametros.f1)
        num_2, den_2 = montar_numden(parametros.b2, parametros.f2)
        num_3, den_3 = montar_numden(parametros.b3, parametros.f3)
        num_4, den_4 = montar_numden(parametros.b4, parametros.f4)
        num_5, den_5 = montar_numden(parametros.b5, parametros.f5)
        num = np.convolve(num_1, num_2)
        den = np.convolve(den_1, den_2)
        num = np.convolve(num, num_3)
        den = np.convolve(den, den_3)
        num = np.convolve(num, num_4)
        den = np.convolve(den, den_4)
        num = np.convolve(num, num_5)
        den = np.convolve(den, den_5)
        return num, den


    def bloco_radiacao():
        num = [1.0, -1.0]
        den = [1.0, 0.0]
        return num, den


    def bloco_ruido():
        num = [1.0, 0.0]
        den = [1.0, -1.0]
        return num, den
```

synth/filtros.py

A.1.3 Fontes

```
    """
    Modulo para geracao das fontes de ruido e de sonorizacao
    """

    import synth.constantes as ctes
    import random as rnd
    import numpy as np
    import pandas as pd
    import synth.utils as utils
```

```
12 def trem_impulsos():
       imp = []
       frequencia_discreta = ctes.Amostragem.TEMPO_AMOSTRAGEM * ctes.
   ParametrosConstantes.F0
       tempo_discreto = int(1/frequencia_discreta)
16     for i in range(ctes.Amostragem.TOTAL_AMOSTRAS):
           imp.append(0.0)
       for i in range(0, ctes.Amostragem.TOTAL_AMOSTRAS, tempo_discreto):
           imp[i] = 1.0
20     return imp


  def trem_pulsos_gloticos():
24     pulsos = []
       periodo_discreto = int(1.0 / (ctes.Amostragem.TEMPO_AMOSTRAGEM *
   ctes.ParametrosConstantes.F0))
       num_pulsos = int(ctes.Amostragem.TOTAL_AMOSTRAS/periodo_discreto)
       for pulso in range(num_pulsos):
28         pulso_glot = pulso_glotico(ctes.Gerais.PORCENTAGEM_GLOTAL, ctes.
   Gerais.K, ctes.Gerais.GANHO_PULSO,
                                      ctes.Gerais.GANHO_RUIDO)
           for amostra in range(len(pulso_glot)):
               pulsos.append(pulso_glot[amostra])
32     return pulsos


  def pulso_glotico(porcentagem_glotal, k_original, ganho_pulso,
   ganho_ruido):
36     """
       Implementado segundo FANT, 1979, Vocal source analysis - a progress
   report
       :param porcentagem_glotal: porcentagem do periodo fundamental que
   forma o periodo do pulso glotico
       :param k_original: parametro do metodo descrito, nos da a queda do
   pulso. será variado neste modulo
40     :param ganho_pulso: controle de ganho do pulso
       :param ganho_ruido: controle de ganho do ruido
       :return: list
       """
44     pulso = []
       f0_variado = ctes.ParametrosConstantes.F0 * rnd.uniform(1.0-ctes.
   Gerais.VARIACAO_F0, 1.0+ctes.Gerais.VARIACAO_F0)
       k = k_original * rnd.uniform(1.0-ctes.Gerais.VARIACAO_K, 1.0+ctes.
   Gerais.VARIACAO_K)

48     tempo_discreto = int(1.0 / (ctes.Amostragem.TEMPO_AMOSTRAGEM *
   f0_variado))
```

```
        wg = ctes.ParametrosConstantes.F0 * 2.0 * np.pi / porcentagem_glotal
        t_subida = int(np.pi * ctes.Amostragem.TAXA_AMOSTRAGEM / wg)
        t_descida = int(((1.0/wg) * np.arccos((k - 1.0) / k)) * ctes.
    Amostragem.TAXA_AMOSTRAGEM)
52      t_vazio = int(tempo_discreto - t_subida - t_descida)

        for i in range(t_subida):
            u = 0.5 * (1.0 - np.cos(wg * i / ctes.Amostragem.TAXA_AMOSTRAGEM
    ))
56          pulso.append(u)
        for i in range(t_descida):
            u = (k * np.cos(wg * i / ctes.Amostragem.TAXA_AMOSTRAGEM) - k +
    1.0)
            pulso.append(u)
60      pulso = utils.modular_escalar(pulso, ganho_pulso)

        ganho_ruido_k = 1
        if k > 1:
64          ganho_ruido_k = 1/k
        ruido = utils.modular_escalar(ruido_gaussiano(t_vazio), ganho_ruido
    * ganho_ruido_k)
        for i in range(t_vazio):
            pulso.append(ruido[i])
68
        return pulso


72  def ruido_branco():
        noise = []
        for i in range(ctes.Amostragem.TOTAL_AMOSTRAS):
            ruido_branco.append(rnd.uniform(0.0, 1.0))
76      return noise


    def ruido_gaussiano(numero_amostras):
80      ruido = list(np.random.normal(ctes.Gerais.CENTRO_RUIDO, ctes.Gerais.
    DESVIO_PADRAO_RUIDO, numero_amostras))
        ruido = utils.normalizar(ruido)
        return ruido


84
    def onda_quadrada():
        sqr = []
        for i in range(ctes.Amostragem.TOTAL_AMOSTRAS):
88          sqr.append(np.sin(2*np.pi*ctes.ParametrosConstantes.F0*i/ctes.
    Amostragem.TAXA_AMOSTRAGEM))
            if sqr[i] >= 0.0:
```

```
                sqr[i] = 1.0
            else:
                sqr[i] = 0.0
        return sqr


def ruido_rosa():
    """
    Implementado de acordo com a descricao em https://www.dsprelated.com
    /showarticle/908.php
    """
    array = np.empty((ctes.Amostragem.TOTAL_AMOSTRAS, ctes.Gerais.
    NUMERO_FONTES_RUIDO_ROSA))
    array.fill(np.nan)
    array[0, :] = np.random.random(ctes.Gerais.NUMERO_FONTES_RUIDO_ROSA)
    array[:, 0] = np.random.random(ctes.Amostragem.TOTAL_AMOSTRAS)

    n = ctes.Gerais.NUMERO_FONTES_RUIDO_ROSA
    cols = np.random.geometric(0.5, n)
    cols[cols >= ctes.Gerais.NUMERO_FONTES_RUIDO_ROSA] = 0
    rows = np.random.randint(ctes.Amostragem.TOTAL_AMOSTRAS, size=n)
    array[rows, cols] = np.random.random(n)

    df = pd.DataFrame(array)
    df.fillna(method='ffill', axis=0, inplace=True)
    return df.sum(axis=1).values


def modulantesenoidal(amostras):
    mod = []
    transicao = int(ctes.Gerais.PORCENTAGEM_MODULACAO_SENOIDAL *
    amostras / 2)
    for i in range(transicao):
        angulo = (np.pi/2.0)*i/transicao
        mod.append(np.sin(angulo))
    for i in range(int(amostras-2*transicao)):
        mod.append(1.0)
    for i in range(transicao):
        angulo = (np.pi / 2.0) * i / transicao
        mod.append(np.cos(angulo))
    return mod
```

synth/fontes.py

A.1.4 Forma de onda

```
"""
```

```
    Modulo para guardar os dados da forma de onda gerada e quando possivel
        adiciona-lo ao arquivo final .wav
    """

    import scipy.io.wavfile as wave
    import synth.constantes as ctes
    import synth.fontes as fontes
    import synth.utils as utils
    import numpy as np


    class Som:

        _arquivo = None
        _valores = None

        def __init__(self, arquivo):
            self._arquivo = arquivo
            self._valores = fontes.trem_pulsos_gloticos()

        @property
        def valores(self):
            return self._valores

        @valores.setter
        def valores(self, value):
            self._valores = value

        def salvararquivo(self):
            wave.write(self._arquivo + '.wav', ctes.Amostragem.
        TAXA_AMOSTRAGEM, np.asarray(self._valores))

        def modular(self, multiplicador):
            for indice in range(len(self._valores)):
                self._valores[indice] = self._valores[indice] *
        multiplicador[indice]

        def inverter(self):
            self._valores = list(reversed(self._valores))

        def normalizar(self):
            self.valores = utils.normalizar(self.valores)

        def somarruido(self, ruido, ganho_ruido):
            """
            Soma ruido aos valores
            :param ruido: list
```

```
        :param ganho_ruido: escalar em dB
        """
48      ganho = 10.0**(ganho_ruido/20.0)
        for i in range(len(self._valores)):
            self._valores[i] += (ruido[i] * ganho)
```

synth/forma_onda.py

A.1.5 Parâmetros

```
1 """
  Definicoes de parametros do sintetizador, geral e generico para varios
     tipo de vozes.
  Os ganhos descritos estão em dB e as frequencias em Hz.
  """
5
  import synth.constantes as ctes

  def setar_ganho_ruido(ganhoRuido):
9     ctes.Gerais.GANHO_RUIDO = ganhoRuido


  def setar_variacao_f0(varF0):
13    ctes.Gerais.VARIACAO_F0 = varF0


  def setar_variacao_k(varK):
17    ctes.Gerais.VARIACAO_K = varK


  def setar_f0(f0):
21    ctes.ParametrosConstantes.F0 = f0


  def zerar_variacao_f0():
25    ctes.Gerais.VARIACAO_F0 = 0


  def zerar_variacao_k():
29    ctes.Gerais.VARIACAO_K = 0


  class ParametrosCascata:
33    # <editor-fold desc="Propriedades">

      _av = 0
      _af = 0
```

```
37      _ah = 0
        _avs = 0

        _fnz = 250
41      _f0 = 80
        _f1 = ctes.ParametrosConstantes.F0
        _f2 = 1450
        _f3 = 2450
45      _f4 = 3300
        _f5 = 3750
        _a2 = 0
        _a3 = 0
49      _a4 = 0
        _a5 = 0
        _a6 = 0
        _ab = 0
53      _b1 = 50
        _b2 = 70
        _b3 = 110
        @property
57      def av(self):
            return self._av

        @property
61      def af(self):
            return self._af

        @property
65      def ah(self):
            return self._ah

        @property
69      def avs(self):
            return self._avs

        @property
73      def fnz(self):
            return self._fnz

        @property
77      def f0(self):
            return self._f0

        @property
81      def f1(self):
            return self._f1
```

```
        @property
85      def f2(self):
            return self._f2

        @property
89      def f3(self):
            return self._f3

        @property
93      def f4(self):
            return self._f4

        @property
97      def f5(self):
            return self._f5

        @property
101     def f6(self):
            return ctes.ParametrosConstantes.F6

        @property
105     def a2(self):
            return self._a2

        @property
109     def a3(self):
            return self._a3

        @property
113     def a4(self):
            return self._a4

        @property
117     def a5(self):
            return self._a5

        @property
121     def a6(self):
            return self._a6

        @property
125     def ab(self):
            return self._ab

        @property
129     def b1(self):
            return self._b1
```

```
        @property
133     def b2(self):
            return self._b2

        @property
137     def b3(self):
            return self._b3

        @property
141     def b4(self):
            return ctes.ParametrosConstantes.B4

        @property
145     def b5(self):
            return ctes.ParametrosConstantes.B5

        @property
149     def b6(self):
            return ctes.ParametrosConstantes.B6

        # </editor-fold>
153
        def __init__(self, vogal, av=None, avs=None):
            if vogal == 'a':
                parametros = ctes.VogalA
157             self._b1 = parametros.B1
                self._b2 = parametros.B2
                self._b3 = parametros.B3
                self._f1 = parametros.F1
161             self._f2 = parametros.F2
                self._f3 = parametros.F3
            self._av = av
            self._avs = avs
```

synth/parametros.py

A.1.6 Sintetizador

```
    """
    Modulo para geracao de som com o sintetizador
    """
 4
    import synth.forma_onda as forma_onda
    import synth.filtros as filtros
    import synth.fontes as fontes
 8  import synth.utils as utils
```

```
def sintetizar(nome_arquivo, vogal):
    som = forma_onda.Som(nome_arquivo)
    utils.plotar(som.valores, 'PULSOS INICIAIS', 'AMOSTRAS', '
    INTENSIDADE')

    filtro_nasal = filtros.FiltroNasal(vogal)
    filtro_formantes = filtros.FiltroFormantes(vogal)
    filtro_radiacao = filtros.FiltroRadiacao()

    som.valores = filtro_nasal.filtrar(som.valores)
    utils.plotar(som.valores, 'PULSOS FILTRO NASAL', 'AMOSTRAS', '
    INTENSIDADE')
    som.valores = filtro_formantes.filtrar(som.valores)
    utils.plotar(som.valores, 'PULSOS FILTRO FORMANTES', 'AMOSTRAS', '
    INTENSIDADE')
    som.valores = filtro_radiacao.filtrar(som.valores)
    utils.plotar(som.valores, 'PULSOS FILTRO RADIACAO', 'AMOSTRAS', '
    INTENSIDADE')

    som.normalizar()

    som.salvararquivo()
    return som
```

synth/sintetizador.py

A.1.7 Utils

```
from scipy import signal
import matplotlib.pyplot as plt
import synth.constantes as ctes
import numpy as np


def bode_numerador_denominador(num, den, titulo):
    filtro = signal.TransferFunction(num, den, dt=ctes.Amostragem.
    TEMPO_AMOSTRAGEM)
    plotar_bode(filtro, titulo)


def plotar_bode(funcao_transferencia, titulo):
    w, mag, phase = funcao_transferencia.bode()
    for i in range(len(w)):
        w[i] /= np.pi * 2000
    f, axarr = plt.subplots(2, sharex=True)
```

```
        axarr[0].semilogx(w, mag)
        axarr[0].set_title('BODE - MAGNITUDE: ' + titulo)
        axarr[0].set_ylabel('dB')
        axarr[1].semilogx(w, phase)
        axarr[1].set_title('BODE - FASE')
        axarr[1].set_xlabel('kHz')
        axarr[1].set_ylabel('Graus')


    def plotar_formaonda(som):
        plt.figure()
        plt.plot(som.valores)
        plt.title('ONDA SONORA OBTIDA')
        plt.xlabel('AMOSTRAS')
        plt.ylabel('INTENSIDADE')


    def plotar(amostras, titulo='', xlabel='', ylabel='', tipo='lin'):
        plt.figure()
        if tipo == 'lin':
            plt.plot(range(len(amostras)), amostras)
        elif tipo == 'sctr':
            plt.scatter(range(len(amostras)), amostras)
        plt.title(titulo)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)


    def mostrar_plots():
        plt.show()


    def maximo_absoluto(vetor):
        maximo = 0
        for i in range(len(vetor)):
            if abs(vetor[i]) > maximo:
                maximo = abs(vetor[i])
        return maximo


    def normalizar(serie):
        norm = []
        maximo = maximo_absoluto(serie)
        for i in range(len(serie)):
            norm.append(serie[i] / maximo)
        return norm
```

```
def plotar_histograma(serie):
    count, bins, ignored = plt.hist(serie, 30, normed=True)
    mu, sigma = ctes.Gerais.CENTRO_RUIDO, ctes.Gerais.
    DESVIO_PADRAO_RUIDO
    plt.plot(bins, 1 (sigma * np.sqrt(2 * np.pi))*np.exp(- (bins - mu)
    **2/(2 * sigma ** 2)), linewidth=2, color='r')


def modular_amplitude(sinal, modulante):
    mod = []
    for i in range(len(sinal)):
        mod.append(sinal[i]*modulante[i])
    return mod


def modular_escalar(sinal, escalar):
    mod = []
    for valor in sinal:
        mod.append(valor*escalar)
    return mod
```

synth/utils.py

A.1.8 Filtros

```
"""
Constantes de uso geral no módulo sintetizador. Os ganhos descritos está
    o em dB e as frequencias em Hz.
"""


class Gerais:
    NUMERO_FONTES_RUIDO_ROSA = 1
    TEMPO_SIMULACAO = 2  # s
    PORCENTAGEM_MODULACAO_SENOIDAL = 0.2
    DESVIO_PADRAO_RUIDO = 0.1
    CENTRO_RUIDO = 0.0
    VARIACAO_F0 = 0.05
    VARIACAO_K = 0.5
    GANHO_RUIDO = 0.5
    PORCENTAGEM_GLOTAL = 0.8
    GANHO_PULSO = 1.0
    K = 1.0


class Amostragem:
```

```
    TAXA_AMOSTRAGEM = 10000  # Hz
    TEMPO_AMOSTRAGEM = 1/TAXA_AMOSTRAGEM  # s
    TOTAL_AMOSTRAS = TAXA_AMOSTRAGEM * Gerais.TEMPO_SIMULACAO


class ParametrosConstantes:
    AN = 0
    A1 = 0
    F6 = 4900
    B4 = 250
    B5 = 200
    B6 = 1000
    FGP = 0
    BGP = 100
    FGZ = 1500
    BGZ = 600
    FNP = 250
    BNP = 100
    BNZ = 100
    FGS = 0
    BGS = 200
    F0 = 150


class VogalA:
    F1 = 620
    F2 = 1220
    F3 = 2550
    B1 = 80
    B2 = 50
    B3 = 140
```

synth/constantes.py

A.2 Códigos da interface gráfica

```
from tkinter import  *
from synth import constantes as ctes
from synth import sintetizador as sintetizador

class Gui:
    def __init__(self, master=None):


        self.fontePadrao = ("Arial", "10")
        self.mainContainer = Frame(master)
```

```
        self.mainContainer["pady"] = 10
        self.mainContainer.pack()

        self.nomeArquivoContainer = Frame(master)
        self.nomeArquivoContainer["padx"] = 20
        self.nomeArquivoContainer.pack()

        self.tempoSimulacaoContainer = Frame(master)
        self.tempoSimulacaoContainer["padx"] = 20
        self.tempoSimulacaoContainer.pack()

        self.variacaoF0Container = Frame(master)
        self.variacaoF0Container["padx"] = 20
        self.variacaoF0Container.pack()

        self.variacaoKContainer = Frame(master)
        self.variacaoKContainer["padx"] = 20
        self.variacaoKContainer.pack()

        self.ganhoRuidoContainer = Frame(master)
        self.ganhoRuidoContainer["padx"] = 20
        self.ganhoRuidoContainer.pack()

        self.ganhoPulsoContainer = Frame(master)
        self.ganhoPulsoContainer["padx"] = 20
        self.ganhoPulsoContainer.pack()

        self.kContainer = Frame(master)
        self.kContainer["padx"] = 20
        self.kContainer.pack()

        self.porcentGlotContainer = Frame(master)
        self.porcentGlotContainer["padx"] = 20
        self.porcentGlotContainer.pack()

        self.anContainer = Frame(master)
        self.anContainer["padx"] = 20
        self.anContainer.pack()

        self.a1Container = Frame(master)
        self.a1Container["padx"] = 20
        self.a1Container.pack()

        self.b4Container = Frame(master)
        self.b4Container["padx"] = 20
        self.b4Container.pack()
```

```
            self.b5Container = Frame(master)
            self.b5Container["padx"] = 20
            self.b5Container.pack()

            self.fnpContainer = Frame(master)
            self.fnpContainer["padx"] = 20
            self.fnpContainer.pack()

            self.bnpContainer = Frame(master)
            self.bnpContainer["padx"] = 20
            self.bnpContainer.pack()

            self.bnzContainer = Frame(master)
            self.bnzContainer["padx"] = 20
            self.bnzContainer.pack()

            self.f0Container = Frame(master)
            self.f0Container["padx"] = 20
            self.f0Container.pack()

            self.f1Container = Frame(master)
            self.f1Container["padx"] = 20
            self.f1Container.pack()

            self.b1Container = Frame(master)
            self.b1Container["padx"] = 20
            self.b1Container.pack()

            self.f2Container = Frame(master)
            self.f2Container["padx"] = 20
            self.f2Container.pack()

            self.b2Container = Frame(master)
            self.b2Container["padx"] = 20
            self.b2Container.pack()

            self.f3Container = Frame(master)
            self.f3Container["padx"] = 20
            self.f3Container.pack()

            self.b3Container = Frame(master)
            self.b3Container["padx"] = 20
            self.b3Container.pack()

            self.sintetizarContainer = Frame(master)
            self.sintetizarContainer["pady"] = 20
            self.sintetizarContainer.pack()
```

```
        self.titulo = Label(self.mainContainer, text="Sintetizador de
    voz")
        self.titulo["font"] = ("Arial", "10", "bold")
        self.titulo.pack()

        self.nomeArquivoLabel = Label(self.nomeArquivoContainer, text="
    Nome do arquivo", font=self.fontePadrao)
        self.nomeArquivoLabel.pack(side=LEFT)
        self.nomeArquivo = Entry(self.nomeArquivoContainer)
        self.nomeArquivo["width"] = 30
        self.nomeArquivo["font"] = self.fontePadrao
        self.nomeArquivo.pack(side=LEFT)

        self.tempoSimulacaoLabel = Label(self.tempoSimulacaoContainer,
    text="Tempo de simulação [s]", font=self.fontePadrao)
        self.tempoSimulacaoLabel.pack(side=LEFT)
        self.tempoSimulacao = Entry(self.tempoSimulacaoContainer,
    textvariable=StringVar(root, value=str(ctes.Gerais.TEMPO_SIMULACAO)))
        self.tempoSimulacao["width"] = 30
        self.tempoSimulacao["font"] = self.fontePadrao
        self.tempoSimulacao.pack(side=LEFT)

        self.variacaoF0Label = Label(self.variacaoF0Container, text="
    Variação máxima de F0 (1 sendo o máximo)",
                                     font=self.fontePadrao)
        self.variacaoF0Label.pack(side=LEFT)
        self.variacaoF0 = Entry(self.variacaoF0Container,
                                textvariable=StringVar(root, value=
    str(ctes.Gerais.VARIACAO_F0)))
        self.variacaoF0["width"] = 30
        self.variacaoF0["font"] = self.fontePadrao
        self.variacaoF0.pack(side=LEFT)

        self.variacaoKLabel = Label(self.variacaoKContainer, text="Varia
    ção máxima de k (1 sendo o máximo)",
                                    font=self.fontePadrao)
        self.variacaoKLabel.pack(side=LEFT)
        self.variacaoK = Entry(self.variacaoKContainer,
                               textvariable=StringVar(root, value=str(
    ctes.Gerais.VARIACAO_K)))
        self.variacaoK["width"] = 30
        self.variacaoK["font"] = self.fontePadrao
        self.variacaoK.pack(side=LEFT)

        self.ganhoRuidoLabel = Label(self.ganhoRuidoContainer, text="
    Ganho do ruído (1 sendo o máximo)",
```

```
                                        font=self.fontePadrao)
144         self.ganhoRuidoLabel.pack(side=LEFT)
            self.ganhoRuido = Entry(self.ganhoRuidoContainer,
                                    textvariable=StringVar(root, value=str(
    ctes.Gerais.GANHO_RUIDO)))
            self.ganhoRuido["width"] = 30
148         self.ganhoRuido["font"] = self.fontePadrao
            self.ganhoRuido.pack(side=LEFT)

            self.anLabel = Label(self.anContainer, text="AN [dB]", font=self
    .fontePadrao)
152         self.anLabel.pack(side=LEFT)
            self.an = Entry(self.anContainer, textvariable= StringVar(root,
    value=str(ctes.ParametrosConstantes.AN)))
            self.an["width"] = 30
            self.an["font"] = self.fontePadrao
156         self.an.pack(side=LEFT)

            self.f1Label = Label(self.f1Container, text="F1 [Hz]", font=self
    .fontePadrao)
            self.f1Label.pack(side=LEFT)
160         self.f1 = Entry(self.f1Container, textvariable=StringVar(root,
    value=str(ctes.VogalA.F1)))
            self.f1["width"] = 30
            self.f1["font"] = self.fontePadrao
            self.f1.pack(side=LEFT)
164
            self.a1Label = Label(self.a1Container, text="A1 [dB]", font=self
    .fontePadrao)
            self.a1Label.pack(side=LEFT)
            self.a1 = Entry(self.a1Container, textvariable=StringVar(root,
    value=str(ctes.ParametrosConstantes.A1)))
168         self.a1["width"] = 30
            self.a1["font"] = self.fontePadrao
            self.a1.pack(side=LEFT)

172         self.b1Label = Label(self.b1Container, text="B1 [Hz]", font=self
    .fontePadrao)
            self.b1Label.pack(side=LEFT)
            self.b1 = Entry(self.b1Container, textvariable=StringVar(root,
    value=str(ctes.VogalA.B1)))
            self.b1["width"] = 30
176         self.b1["font"] = self.fontePadrao
            self.b1.pack(side=LEFT)

            self.b4Label = Label(self.b4Container, text="B4 [Hz]", font=self
    .fontePadrao)
```

```
180         self.b4Label.pack(side=LEFT)
            self.b4 = Entry(self.b4Container, textvariable=StringVar(root,
    value=str(ctes.ParametrosConstantes.B4)))
            self.b4["width"] = 30
            self.b4["font"] = self.fontePadrao
184         self.b4.pack(side=LEFT)

            self.f2Label = Label(self.f2Container, text="F2 [Hz]", font=self
    .fontePadrao)
            self.f2Label.pack(side=LEFT)
188         self.f2 = Entry(self.f2Container, textvariable=StringVar(root,
    value=str(ctes.VogalA.F2)))
            self.f2["width"] = 30
            self.f2["font"] = self.fontePadrao
            self.f2.pack(side=LEFT)
192
            self.b5Label = Label(self.b5Container, text="B5 [Hz]", font=self
    .fontePadrao)
            self.b5Label.pack(side=LEFT)
            self.b5 = Entry(self.b5Container, textvariable=StringVar(root,
    value=str(ctes.ParametrosConstantes.B5)))
196         self.b5["width"] = 30
            self.b5["font"] = self.fontePadrao
            self.b5.pack(side=LEFT)

200         self.fnpLabel = Label(self.fnpContainer, text="FNP [Hz]", font=
    self.fontePadrao)
            self.fnpLabel.pack(side=LEFT)
            self.fnp = Entry(self.fnpContainer, textvariable=StringVar(root,
     value=str(ctes.ParametrosConstantes.FNP)))
            self.fnp["width"] = 30
204         self.fnp["font"] = self.fontePadrao
            self.fnp.pack(side=LEFT)

            self.bnpLabel = Label(self.bnpContainer, text="BNP [Hz]", font=
    self.fontePadrao)
208         self.bnpLabel.pack(side=LEFT)
            self.bnp = Entry(self.bnpContainer, textvariable=StringVar(root,
     value=str(ctes.ParametrosConstantes.FNP)))
            self.bnp["width"] = 30
            self.bnp["font"] = self.fontePadrao
212         self.bnp.pack(side=LEFT)

            self.bnzLabel = Label(self.bnzContainer, text="BNZ [Hz]", font=
    self.fontePadrao)
            self.bnzLabel.pack(side=LEFT)
```

```
216         self.bnz = Entry(self.bnzContainer, textvariable=StringVar(root,
        value=str(ctes.ParametrosConstantes.BNZ)))
            self.bnz["width"] = 30
            self.bnz["font"] = self.fontePadrao
            self.bnz.pack(side=LEFT)
220
            self.f0Label = Label(self.f0Container, text="F0 [Hz]", font=self
        .fontePadrao)
            self.f0Label.pack(side=LEFT)
            self.f0 = Entry(self.f0Container, textvariable= StringVar(root,
        value=str(ctes.ParametrosConstantes.F0)))
224         self.f0["width"] = 30
            self.f0["font"] = self.fontePadrao
            self.f0.pack(side=LEFT)

228         self.b2Label = Label(self.b2Container, text="B2 [Hz]", font=self
        .fontePadrao)
            self.b2Label.pack(side=LEFT)
            self.b2 = Entry(self.b2Container, textvariable=StringVar(root,
        value=str(ctes.VogalA.B2)))
            self.b2["width"] = 30
232         self.b2["font"] = self.fontePadrao
            self.b2.pack(side=LEFT)

            self.f3Label = Label(self.f3Container, text="F3 [Hz]", font=self
        .fontePadrao)
236         self.f3Label.pack(side=LEFT)
            self.f3 = Entry(self.f3Container, textvariable=StringVar(root,
        value=str(ctes.VogalA.F3)))
            self.f3["width"] = 30
            self.f3["font"] = self.fontePadrao
240         self.f3.pack(side=LEFT)

            self.b3Label = Label(self.b3Container, text="B3 [Hz]", font=self
        .fontePadrao)
            self.b3Label.pack(side=LEFT)
244         self.b3 = Entry(self.b3Container, textvariable=StringVar(root,
        value=str(ctes.VogalA.B3)))
            self.b3["width"] = 30
            self.b3["font"] = self.fontePadrao
            self.b3.pack(side=LEFT)
248
            self.kLabel = Label(self.kContainer, text="k", font=self.
        fontePadrao)
            self.kLabel.pack(side=LEFT)
            self.k = Entry(self.kContainer, textvariable=StringVar(root,
        value=str(ctes.Gerais.K)))
```

```
252         self.k["width"] = 30
            self.k["font"] = self.fontePadrao
            self.k.pack(side=LEFT)

256         self.percentGlotLabel = Label(self.porcentGlotContainer, text="
    Relaçao FG/F0", font=self.fontePadrao)
            self.percentGlotLabel.pack(side=LEFT)
            self.percentGlot = Entry(self.porcentGlotContainer, textvariable
    =StringVar(root, value=str(ctes.Gerais.PORCENTAGEM_GLOTAL)))
            self.percentGlot["width"] = 30
260         self.percentGlot["font"] = self.fontePadrao
            self.percentGlot.pack(side=LEFT)

            self.ganhoPulsoLabel = Label(self.ganhoPulsoContainer, text="
    Ganho do pulso (1 sendo o máximo)", font=self.fontePadrao)
264         self.ganhoPulsoLabel.pack(side=LEFT)
            self.ganhoPulso = Entry(self.ganhoPulsoContainer, textvariable=
    StringVar(root, value=str(ctes.Gerais.K)))
            self.ganhoPulso["width"] = 30
            self.ganhoPulso["font"] = self.fontePadrao
268         self.ganhoPulso.pack(side=LEFT)

            self.autenticar = Button(self.sintetizarContainer)
            self.autenticar["text"] = "Sintetizar"
272         self.autenticar["font"] = ("Calibri", "8")
            self.autenticar["width"] = 12
            self.autenticar["command"] = self.sintetizar
            self.autenticar.pack()
276         self.mensagem = Label(self.sintetizarContainer, text="", font=
    self.fontePadrao)
            self.mensagem.pack()

        # Listener para sintetizar
280     def sintetizar(self):
            nomeArquivo = self.nomeArquivo.get()
            try:
                if nomeArquivo != "":
284                 self.mensagem["text"] = "Sintetizando"
                    ctes.Gerais.TEMPO_SIMULACAO = float(self.tempoSimulacao.
    get())
                    ctes.Gerais.VARIACAO_F0 = float(self.variacaoF0.get())
                    ctes.Gerais.VARIACAO_K = float(self.variacaoK.get())
288                 ctes.Gerais.GANHO_RUIDO = float(self.ganhoRuido.get())
                    ctes.Gerais.K = float(self.k.get())
                    ctes.Gerais.GANHO_PULSO = float(self.ganhoPulso.get())
                    ctes.Gerais.PORCENTAGEM_GLOTAL = float(self.percentGlot.
    get())
```

```
292                     ctes.ParametrosConstantes.F0 = float(self.f0.get())
                        ctes.ParametrosConstantes.B4 = float(self.b4.get())
                        ctes.ParametrosConstantes.B5 = float(self.b5.get())
                        ctes.ParametrosConstantes.FGP = float(self.fgp.get())
296                     ctes.ParametrosConstantes.BGP = float(self.bgp.get())
                        ctes.ParametrosConstantes.FGZ = float(self.fgz.get())
                        ctes.ParametrosConstantes.BGZ = float(self.bgz.get())
                        ctes.ParametrosConstantes.FNP = float(self.fnp.get())
300                     ctes.ParametrosConstantes.BNP = float(self.bnp.get())
                        ctes.ParametrosConstantes.BNZ = float(self.bnz.get())
                        ctes.ParametrosConstantes.FGS = float(self.fgs.get())
                        ctes.ParametrosConstantes.BGS = float(self.bgs.get())
304                     ctes.VogalA.F1 = float(self.f1.get())
                        ctes.VogalA.B1 = float(self.b1.get())
                        ctes.VogalA.F2 = float(self.f2.get())
                        ctes.VogalA.B2 = float(self.b2.get())
308                     ctes.VogalA.F3 = float(self.f3.get())
                        ctes.VogalA.B3 = float(self.b3.get())
                        sintetizador.sintetizar(nomeArquivo, 'a')
                        self.mensagem["text"] = "Áudio sintetizado!"
312                 else:
                        self.mensagem["text"] = "Nome de arquivo vazio"
            except:
                self.mensagem["text"] = "Erro: " + sys.exc_info()[0]
316
    if __name__ == "__main__":
        root = Tk()
        root.title('Sintetizador de voz')
320     Gui(root)
        root.mainloop()
```

gui/gui.py

APÊNDICE B – Amostras geradas

As amostras geradas com este trabalho podem ser acessadas pelo link: <https://drive.google.com/open?id=0BwivABSTnP4kNXQ4WVRjajZOeFE>, acesso em 4 de jul. de 2017.

Os parâmetros utilizados para gerar estas amostras podem ser encontrados na tabela 1.