Bruno Franceschini Canale

SÍNTESE DE SOFTWARE PARA PROJETO DE ENLACES DE RÁDIO DIGITAL PONTO-A-PONTO.

Trabalho de Conclusão de Curso do aluno Bruno Franceschini Canale, do curso de Engenharia Elétrica com ênfase em eletrônica da Escola de Engenharia de São Carlos, Universidade de São Paulo.

Orientador: Amilcar Careli Cesar

São Carlos
2015

Franceschini Canale, Bruno
F212s Síntese de software para projeto de enlaces de rádio digital ponto-a-ponto / Bruno Franceschini Canale; orientador Amílcar Careli César. São Carlos, 2015.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Matlab. 2. enlace de rádio. 3. rádio digital. 4. programa rádio. I. Título.

FOLHA DE APROVAÇÃO

Nome: Bruno Franceschini Canale

Título: "Síntese de software para projeto de enlaces de rádio digital ponto-a-ponto"

Trabalho de Conclusão de Curso defendido e aprovado em 30/11/2015,

com NOTA 10 (dez), **pela Comissão Julgadora:**

Prof. Titular Amílcar Careli César - (Orientador - SEL/EESC/USP)

Mestre Arturo Miranda Vera - (Doutorando - SEL/EESC/USP)

Mestre Rafael Jales Lima Ferreira - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

AGRADECIMENTOS

Agradeço à minha família, especialmente aos meus pais, os quais não medem esforços para auxiliar no sucesso de seus filhos.

Agradeço aos meus amigos da USP, os quais fizeram de minha graduação, muito mais que um diploma.

Agradeço a Bati, Biga, Cão, Diogo, Gigli, Romeu e Zé, por me ouvirem falar tanto de antenas.

Agradeço a empresa OPENCADD Advanced Technology, por fornecer toda a infra-estrutura para a realização do trabalho, fora as risadas.

Agradeço a Mário Franceschini, o pioneiro.

RESUMO

CANALE, B. F. (2015). ***Síntese de software para projeto de enlaces rádio digital ponto-a-ponto.*** Trabalho de Conclusão de Curso (Graduação) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015

Um enlace de rádio consiste na comunicação entre dois pontos por meio de ondas eletromagnéticas de alta frequência. Em enlaces de comunicação digital, as portadoras transportam bits que representam a informação. Para que esta comunicação ocorra com qualidade, faz-se necessário estudar e prever as atenuações, ruídos, interferências e obstáculos geográficos, os quais degradam a qualidade do sinal. A relação entre os diversos fatores exige interatividade e tornam o projeto demorado se realizado manualmente. Portanto, o trabalho em questão propõe e concebe uma ferramenta computacional interativa, na qual se pode projetar um enlace de rádio, a partir das coordenadas geográficas dos pontos comunicantes e de outros parâmetros técnicos, como ganho das antenas, potência de transmissão, sensibilidade do receptor, método de modulação e outros, contabilizando os fatores preponderantes na degradação da portadora. O programa fornece um estudo do terreno do enlace, com a visualização da região com uma imagem em satélite, e seu respectivo perfil. Também inclui modulações digitais de fase e amplitude, indicando a performance do sistema de acordo com os parâmetros. Com o auxílio desta ferramenta, o usuário pode alterar os parâmetros até encontrar um projeto otimizado que satisfaça as exigências. O programa é desenvolvido em linguagem *Matlab* para o sistema operacional *Windows*.

Palavras-Chave: Matlab, enlace de rádio, rádio digital, programa rádio

ABSTRACT

CANALE, B. F. (2015). ***Synthesis of a software to calculate digital point-to-point radio link budget***. Trabalho de Conclusão de Curso (Graduação) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015

A radio link consists essentially in a communication between two points with high frequency electromagnetic waves. These waves, which are called carriers, are emitted and captured by antennas, and travel through the air carrying information. In digital radio links, the carriers carry the bits, which represents the transmitted information. To guarantee the quality of the communication, there´s a need to study and foresee signal attenuations, noise, interference and geographical obstacles, which degrade the carriers along the way. Since these factors are strongly related, a manual design would be a very difficult task, and an interactive approach becomes interesting. Therefore, this academic work suggests and conceives a computational tool, in which the user can design a radio link, from geographical coordinates of the communication points and other technical parameters like antenna gains, transmission power, receiver´s sensibility, modulation method and others, considering the major damaging factors for carrier waves. The software studies the radio link terrain, by a satellite visualization and a landscape profile. It also includes phase and amplitude modulation techniques, indicating the performance due to the parameters choices. With this tool, the user can tune the parameters until all the link requisites are satisfied. The software was developed in Matlab for Windows operational system.

Key Words: Matlab, radio link, digital radio, software radio.

Sumário

1. INTRODUÇÃO

Neste capítulo, encontram-se a apresentação do trabalho, e as motivações e objetivos que culminaram na sua síntese.

1.1 Apresentação

Este trabalho consiste no desenvolvimento completo de um *software* para cálculo de parâmetros técnicos de enlaces digitais de rádio entre dois pontos quaisquer no globo terrestre, a partir das coordenadas geográficas. Mais especificamente, o *software* calcula atenuações de percurso; fornece a visualização do terreno e do plano planaltimétrico do enlace, acusando possíveis obstruções da visada direta entre as antenas ou da primeira Zona de Fresnel; permite ao usuário observar de forma dinâmica os resultados das especificações técnicas do enlace, ao manipular o ganho das antenas, suas respectivas alturas de instalação, potência de transmissão, sensibilidade de recepção, e demais parâmetros; mostra o desempenho do sistema com os principais modos de modulação digital e exporta os dados em formato *html*.

Este documento está estruturado da seguinte forma. Primeiramente, faz-se esta apresentação para contextualizar o leitor, seguido da motivação e objetivos que culminaram no trabalho. Depois, introduz-se o programa, mostrando o layout, as funções, os botões, os textos e outras especificações de interface, para em seguida exibir os materiais e métodos envolvidos.

Após concretizada a familiarização do leitor com o programa, detalham-se as funcionalidades do *software*, uma a uma, a partir de um prelúdio teórico e da forma como fora implementada. Em seguida, valida-se o programa, comparando os cálculos e gráficos com ferramentas conhecidas do mercado. Finalmente, o trabalho se despede com a discussão dos resultados obtidos, e uma breve conclusão, discutindo-se o sucesso do programa perante seus objetivos.

Em relação às ferramentas para validação, é justo afirmar que serviram também de inspiração para o desenvolvimento do *software*, desde o *layout* até as funcionalidades a serem implementadas. Os dois programas que desempenharam este papel fundamental na síntese do trabalho foram: *RF Haversinke Lite* da empresa *Cristinel Bontas © 2013 Tony Mocanu & Cristinel Bontas,* e *PathCheck™* da empresa *Trango Systems.*

1.2 Motivação e Objetivos

A motivação para a concepção deste programa está na síntese de uma ferramenta gratuita que auxilie nos projetos de enlaces de rádio, e que também estimule o desenvolvimento de novas tecnologias e conhecimentos desta área. Por ser um programa aberto, oferece espaço para o debate dos conhecimentos que o permeiam, e das técnicas envolvidas na concepção de enlaces. A motivação também tem um caráter pedagógico, já que permite visualizar como ocorrem as interações entre os parâmetros técnicos em um enlace, e como eles impactam no desempenho. Durante a realização do programa, aprende-se muito sobre *projeto* de *software,* criando assim uma motivação adicional, a de também compartilhar este conhecimento. Como há intenção de compartilhar o programa, decidiu-se por realizar o *software* em língua inglesa. Assim, resumem-se os objetivos da seguinte maneira:

Desenvolver um programa com código-aberto de cálculo de parâmetros técnicos para um enlace qualquer de radiofrequência, para posterior divulgação e compartilhamento, estimulando assim o aprendizado e o desenvolvimento dos conhecimentos que o compõem, tanto na área de concepção de *software*, quanto em telecomunicações. Portanto, cabe aqui descrever tanto os códigos que compreendem o *software*, quanto as funções relacionadas a telecomunicações.

2. IMPLEMENTAÇÃO

Neste capítulo será explanado como foram implementados os diversos cálculos e análises que fazem parte deste *software*. Porém, primeiramente mostra-se o programa, explicando seu funcionamento e, em seguida, detalham-se as funcionalidades por temas, da forma que foram organizadas no programa

2.1 O programa

A versão final do *software* apresenta o seguinte aspecto (Fig. 1):

Figura 1 – Versão final do programa.. Fonte: autor.

Como se pode ver, o programa está assim organizado, tematicamente:

- *Geographical Coordinates* (Coordenadas Geográficas):

  Local em que o usuário deve informar as latitudes e longitudes dos pontos nos quais se deseja calcular o enlace de rádio.

- *Link Parameters* (Parâmetros do Enlace):

  Entradas relacionadas ao enlace. O usuário deve fornecer a frequência da portadora, largura de banda, uma margem de segurança para o enlace, o número de amostras da representação gráfica e o tipo de polarização utilizada.

- *Tx Characteristics* (Características do Transmissor):

  Parte do *software* dedicada às características do transmissor. O usuário deve fornecer potência, ganho, altura, e elevação em graus da antena.

- *Rx Characteristics* (Características do Receptor):

  Parte do *software* dedicada às características do receptor. O usuário deve fornecer a sensibilidade, ganho, altura, temperatura de ruído da antena, e a figura de ruído do receptor.

- *Attenuations* (Atenuações):

  Local onde se encontram os cálculos de atenuações de percurso. Mais especificamente, calculam-se as atenuações do espaço livre e de chuva.

- *Performance per Modulation* (Desempenho por Modulação):

  Parte do software dedicada ao método de modulação digital. O usuário pode permutar entre os métodos que deseja utilizar e, assim, comparar o desempenho em BER *(Bit-Error Rate)*

- *Data* (Manipulação dos dados)

  Região do software em que o usuário pode atualizar os dados, caso alguma entrada tenha sido alterada, e exportá-los em um arquivo de extensão *html* como forma de documentar o projeto do enlace.

O programa ainda contém, na sua parte lateral direita, uma área para a visualização dos gráficos referentes ao terreno estudado e uma caixa de texto contendo considerações técnicas do enlace, como altura dos pontos de transmissão e recepção, potência recebida no receptor, desempenho do sistema e outros.

O *software* funciona da seguinte forma. O usuário deve obrigatoriamente fornecer as entradas já antecipadas anteriormente, clicar em *Update Data* para calcular os parâmetros e executar os gráficos. É importante ressaltar que este botão é o elemento responsável pelo

dinamismo do *software*. A partir dele, toda vez que o usuário mudar um parâmetro do enlace deve apertar o botão para observar os resultados em atenuação e desempenho. Desta forma, pode-se facilmente encontrar uma configuração interessante ao usuário, mudando os parâmetros e observando as mudanças. Caso se queira salvar as características calculadas, o usuário pode clicar em *Export Data* a qualquer momento, como já dito anteriormente, salvando-as em um arquivo em *html*. Para se permutar entre os gráficos, clica-se no botão *Plot Planal/Plot Google*.

2.2 Materiais e Métodos

O programa foi totalmente desenvolvido em *Matlab*, desde a interface gráfica, cálculos, gerenciamento dos arquivos, comunicações com API (*Application* Programming *Interface*) e o executável do *software*. Optou-se por esta linguagem, exatamente por apresentar todos os instrumentos necessários para sintetizar uma ferramenta de engenharia, desde sua idealização até sua conclusão, em um ambiente de programação de alto nível. As bibliotecas e aplicativos usados são descritas a seguir:

- *Matlab 2015a*:

  Ambiente de desenvolvimento em que se utilizam as demais bibliotecas e aplicativos. Também possui biblioteca própria, e foram usadas para as operações matemáticas e para a comunicação com os APIs.

- *Communication Systems Toolbox 5.7*:

  Esta biblioteca apresenta funções para analisar, projetar e verificar sistemas de comunicação. No programa em questão, foram usadas as funções que calculam desempenho de modulação digital *(Bit Error Rate).*

- *Phased-Array Toolbox 2.3*:

  Biblioteca responsável por aplicações de radar, sonar e comunicações sem fio. Foram usadas apenas a função de atenuação de percurso e as constantes físicas desta ferramenta.

- *MATLAB Compiler 5.2*:

  Este aplicativo tem como função gerar aplicações *stand-alone* dos códigos desenvolvidos em *Matlab*. Em outras palavras, esta ferramenta permite exportar para não usuários de *Matlab* um código desenvolvido nesta plataforma. Naturalmente, este aplicativo foi usado para tornar o software independente do seu ambiente de criação.

- *MATLAB GUI (Grafical User Interface):*

Esta biblioteca fornece funções para desenvolver ambientes virtuais. Ele permitiu a interação entre o programa e o usuário.

- *MATLAB GUIDE (Graphical User Interface Design Environment):*

  Aplicativo para gerar as figuras, botões, textos e outros utensílios gráficos que compõem o *layout* de um programa. Foi útil na parte gráfica do *software*.

- *MATLAB Report Generator 4.0:*

  Este aplicativo consiste em uma ferramenta de documentação personalizável, que usa uma linguagem própria e se comunica com o ambiente de desenvolvimento do *Matlab*. Portanto, é possível criar modelos de documentação particulares às aplicações de interesse do usuário. Este utensílio foi usado para exportar os dados do enlace em extensão *html*, com uma tabela contendo os parâmetros técnicos, os gráficos referentes ao terreno, e um capítulo dedicado às considerações de cunho técnico.

- *Simulink Project*:

  Este aplicativo é um gerenciador de arquivos para a criação de *softwares* e aplicativos. Ele possui diversas funcionalidades, como identificar arquivos modificados, discriminando a data, local, pessoa que o modificou e o que foi modificado; faz análises de dependências entre os arquivos, acusando possíveis variáveis ou arquivos não referenciados; controle de versões do *software*; controle de repositório e outras. Para a realização deste trabalho, esta ferramenta foi usada principalmente para se ter controle das alterações dos arquivos, mantendo sempre uma versão anterior, para resgatá-la caso houvesse problemas no código que não pudessem ser identificados e solucionados. Esta metodologia de gerenciamento de arquivos já está bem difundida na indústria, e optou-se por utilizá-la não apenas por seus benefícios já citados, mas para proporcionar um caráter profissional na síntese do *software*.

A realização do programa seguiu a seguinte metodologia. Implementou-se primeiramente as funcionalidades consideradas primordiais a uma ferramenta desta natureza. Em seguida, pesaram-se as demais possíveis funcionalidades segundo viabilidade técnica e tempo de execução. Portanto, cronologicamente, em um primeiro momento, focou-se em realizar o estudo do relevo a partir das coordenadas geográficas, desenvolvendo os códigos responsáveis por comunicar o programa com as ferramentas Google, para assim realizar o gráfico planaltimétrico e de visualização de satélite do enlace. Em seguida, foram implementados os cálculos técnicos de enlaces de rádio, relacionados às antenas, atenuações e potências de transmissão e recepção.

Após validadas estas tarefas, as quais foram consideradas indispensáveis para um programa desta natureza, pensou-se em outras funcionalidades interessantes a serem implementadas. Seguindo o critério já citado, optou-se por implementar os cálculos de desempenho das modulações digitais.

Finalmente, como anteriormente não havia forma de salvar os dados calculados, evidenciou-se a necessidade de exportá-los. Portanto, como implementação final, escolheu-se realizar a função de gerar um arquivo *html* do cálculo do enlace.

A validação dos cálculos foi feita seguindo sempre os dois programas de referência já citados na introdução. Um capítulo será inteiramente dedicado a provar a validade da ferramenta posteriormente neste trabalho.

2.3 Interface Gráfica

Como já descrito anteriormente, os elementos gráficos foram criados pela ferramenta GUIDE e suas interações com o usuário foram feitas a partir de um GUI do *Matlab*. Isto foi feito da seguinte maneira:

Ao se criar uma *Graphical User Interface* no Matlab, cria-se automaticamente dois arquivos, um com extensão *fig* e outro com extensão *m*, ambos com o mesmo nome. No caso do desenvolvimento deste software, os arquivos receberam o nome de *layout_tcc*. Para se entender o intuito do arquivo de extensão *m*, faz-se necessário explanar primeiramente o arquivo com extensão *fig*, o qual representa a interface gráfica do programa, sendo esta o ambiente virtual em que o desenvolvedor poderá adicionar botões, textos, imagens, e outros elementos gráficos.

A Fig. 2 mostra parcialmente o arquivo *fig* do programa, quando ainda estava na fase de desenvolvimento, aberto no aplicativo GUIDE.

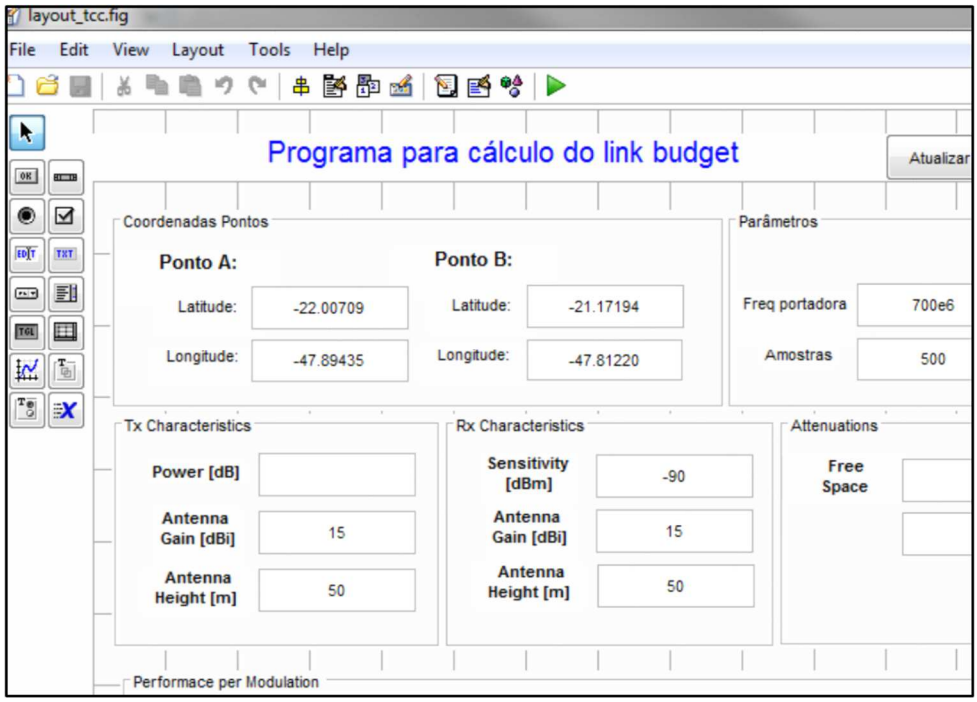


Figura 2 – Imagem que mostra a ferramenta GUIDE em funcionamento, com o arquivo de interface do programa. Fonte: autor.

Pode-se ver na imagem, no canto superior esquerdo, uma paleta de opções gráficas (botões, textos estáticos, caixas de texto, painéis, gráficos e outros). A partir dela, criaram-se os elementos gráficos que compõem a interface.

Com esta ferramenta, toda vez que se adiciona um elemento novo, geram-se automaticamente as funções responsáveis pela interação deste elemento com o usuário. Estas funções são todas declaradas e agrupadas no arquivo de extensão *m*. Portanto, no caso do desenvolvimento do software em questão, o ambiente virtual é composto dos arquivos:

- *Layout_tcc.fig*:

  Figura que representa a interface gráfica do programa.

A Fig. 3 mostra o programa aberto na ferramenta GUIDE.

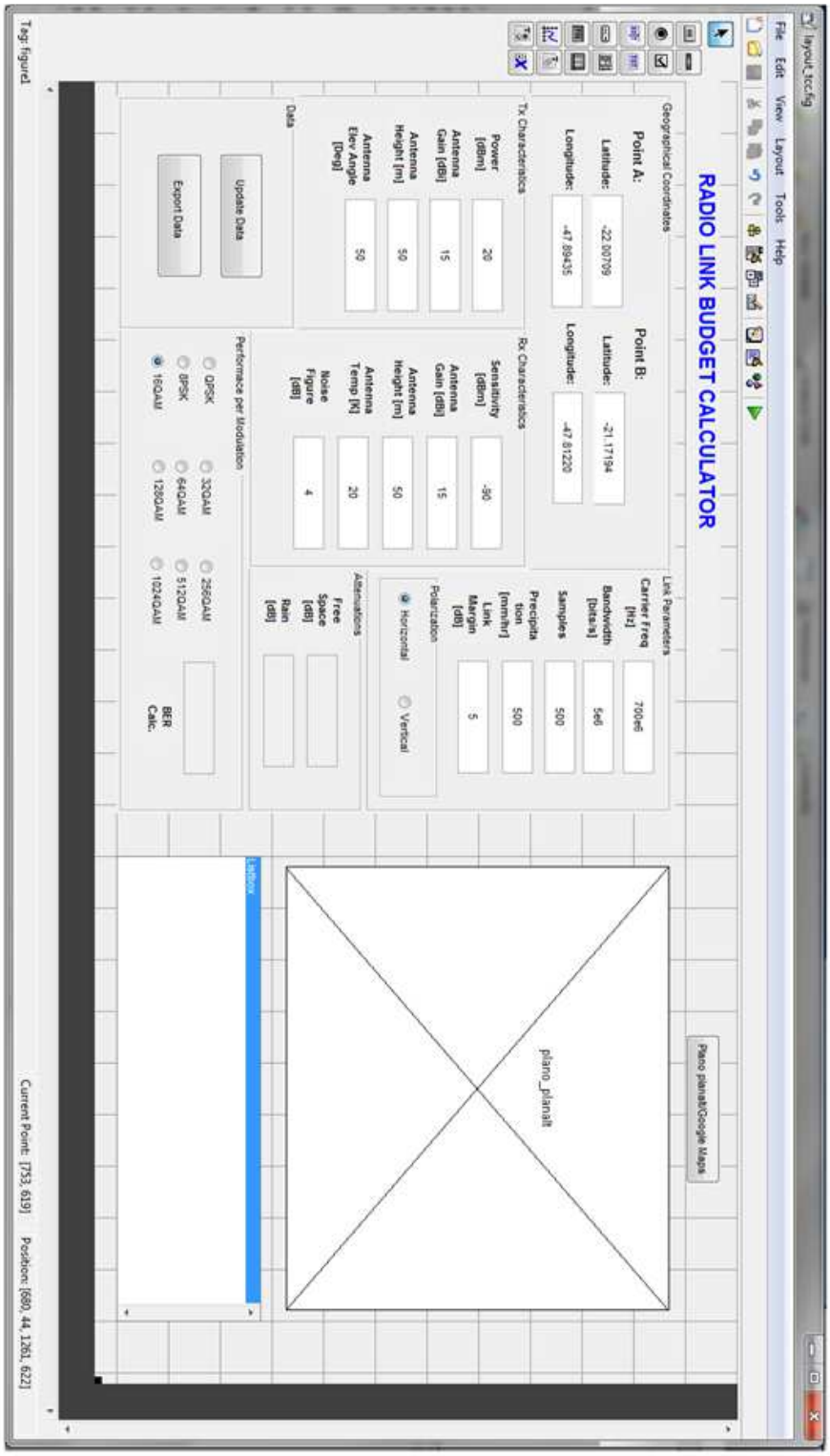


Figura 3 – Figura do programa aberta no aplicativo GUIDE.

- *Layout_tcc.m*:

  Biblioteca com as funções de interação dos elementos gráficos da interface com o usuário. Este arquivo está no Anexo 1.

Como pode-se observar ao analisar o arquivo *o* Anexo 1, as funções de interação se resumem nas categorias:

- *CreateFcn*:

  Função que é executada quando o elemento gráfico é criado.

- *OpeningFcn*:

  Função que é executada quando o elemento gráfico é iniciado.

- *CallbackFcn*:

  Esta função é chamada toda vez que o usuário interage com o elemento em questão. Portanto, para um botão, sua função de *callbac*k é executada quando ele é clicado pelo usuário.

Para melhor compreender como as funções são referenciadas aos elementos, e como são utilizadas dentro do programa, faz-se necessária uma breve discussão sobre a natureza destes elementos, como foram usados, estruturados e organizados na figura de interface.

A Fig. 4 mostra os elementos usados no software.

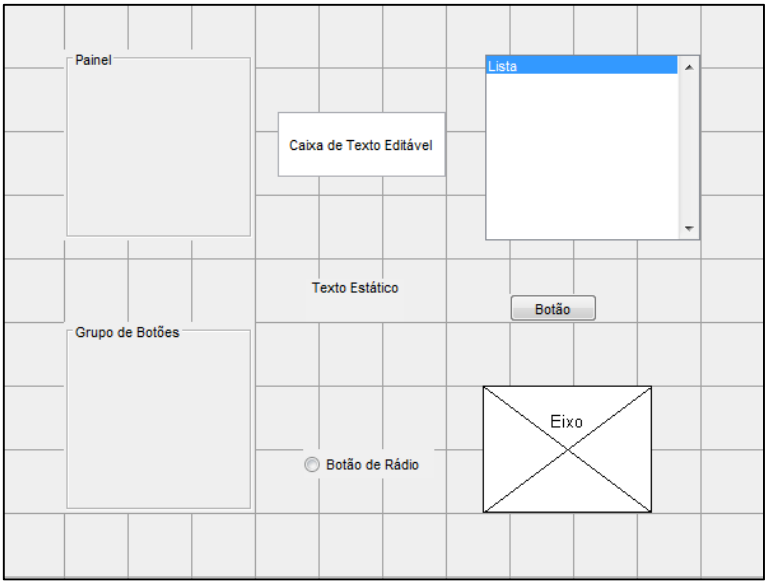


Figura 4 – Demonstração dos elementos gráficos usados no software. Fonte: autor.

Para se entender a aplicação desses utensílios dentro do software, são feitas as seguintes considerações:

- Botão:

  Para esta ferramenta gráfica, foram usadas as funções de *callback* para realizar os cálculos, exportar os dados, e realizar a permutação dos gráficos.

- Caixa de Texto Editável

  Ferramenta gráfica usada para os dados de entrada ou saída do programa, ou seja, para a visualização dos cálculos realizados, e para a entrada das características técnicas do enlace fornecidas pelo usuário. Para estes elementos, não foram usadas suas funções de interação.

- Grupo de Botões:

  Este painel é usado em conjunto com o elemento Botão de Rádio, quando se quer oferecer opções para o usuário. Ao agrupar diversos Botões de Rádio dentro de um grupo, é obrigatória a seleção de apenas um deles. Este conjunto de ferramentas foi claramente usado para a seleção do esquema de modulação digital, no grupo denominado *Perfomance per Modulation*. Outro conjunto também foi utilizado para escolher a polarização da transmissão do enlace, dentro do painel *Polarization.* A partir do painel, pode-se saber qual "Botão de Rádio" está selecionado e, assim descobrir qual opção foi escolhida pelo usuário. As funções de interação dos grupos não foram usadas, e as informações coletadas a partir delas foram feitas diretamente em sua estrutura. Esta forma de coleta será explanada posteriormente ainda nesta subseção.

- Botões de Rádio

  Estes utensílios funcionam como variáveis booleanas, ou seja, podem estar selecionados ou não. Sua forma de implementação mais corrente já foi detalhada no ponto anterior. Foram usadas as funções de *callback* destes elementos no painel *Performance per Modulation*, para permutar o desempenho visualizado em *BER Performance* quando outro esquema de modulação é selecionado pelo usuário. Naturalmente, a função de *callback* de um botão de rádio é invocada quando o mesmo é selecionado.

- Texto Estático

  Elementos usados para introduzir texto no programa, que não possuem quase nenhuma possibilidade de interação com o usuário. Foram usados muitos textos estáticos no *software* para informar ao usuário do que se trata cada caixa de texto

editável. Suas funções de interação se resumem às relacionadas à criação e inicialização do componente e, assim, não foram usadas pelo desenvolvedor.

- Painel

  Apesar de ter sua aparência idêntica ao elemento Grupo de Botões, o mesmo não apresenta as mesmas características. Basicamente, a função desempenhada por este elemento no programa foi puramente estética, organizando os textos estáticos e caixas de texto editáveis por tema. Como esperado, suas funções de interação não foram utilizadas.

- Eixo

  Este elemento é geralmente responsável por abrigar figuras externas ou gráficos. No programa em questão, foram usados dois eixos para gerar os gráficos referentes ao terreno do enlace. A forma com que se interagiu com este utensílio foi a partir de sua estrutura e esta metodologia será explanada a seguir.

- Lista

  Elemento no qual se pode criar listas e identificar qual opção está sendo selecionada pelo usuário. No programa este utensílio foi utilizado simplesmente para abrigar um texto, e não se utilizou desta funcionalidade de seleção.

É importante ressaltar que ao se dizer que “as funções de interação não foram usadas”, não significa que elas não estão declaradas na biblioteca. De fato, ao se observar o Anexo 1, podem-se ver todas as funções de criação e inicialização dos elementos gráficos. À vista disso, esta frase apenas informa que o desenvolvedor não adicionou código às funções de interação.

Elucidada a forma em que os elementos foram empregados no programa, é relevante explanar sobre suas estruturas.

2.3.1 Estrutura Gráfica

Em *Matlab*, uma interface gráfica de usuário (*Graphical User Interface*) é uma estrutura que remete a uma *struct* de objetos, e que leva o nome de *handles*.

A Fig. 5 mostra parcialmente os objetos contidos no *handles* do software.

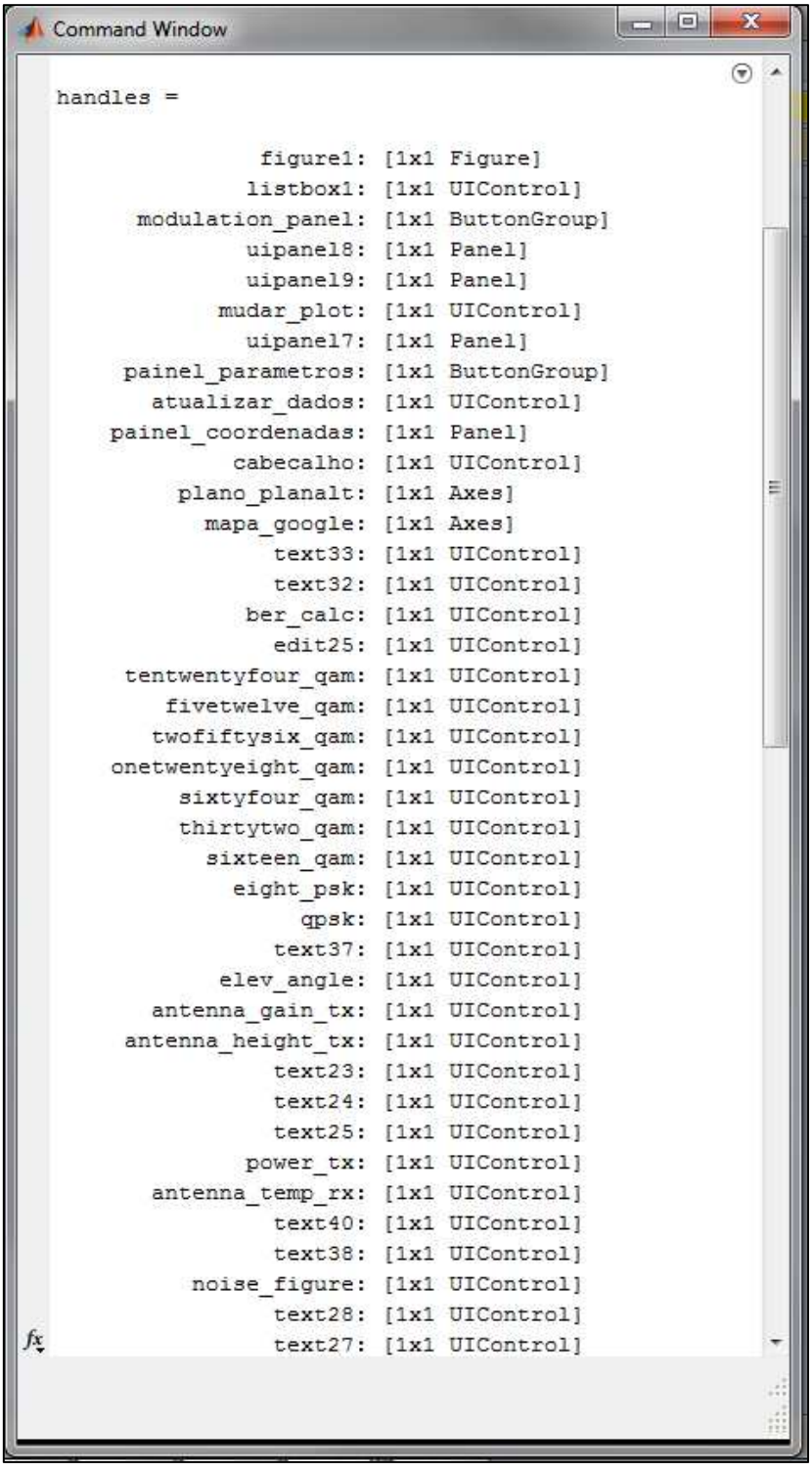


Figura 5 – Objetos contidos no *handles* do GUI do *software*. Fonte: autor.

Como se pode ver na figura, o *handles* contêm a figura de interface (no caso o objeto nomeado de *figure1*) e todos os elementos gráficos. Estes objetos também possuem uma estrutura própria com um formato que também remete a uma variável do tipo *struct.*

A Fig. 6 exemplifica como é a estrutura de um botão, no caso, o botão *Update Data*.

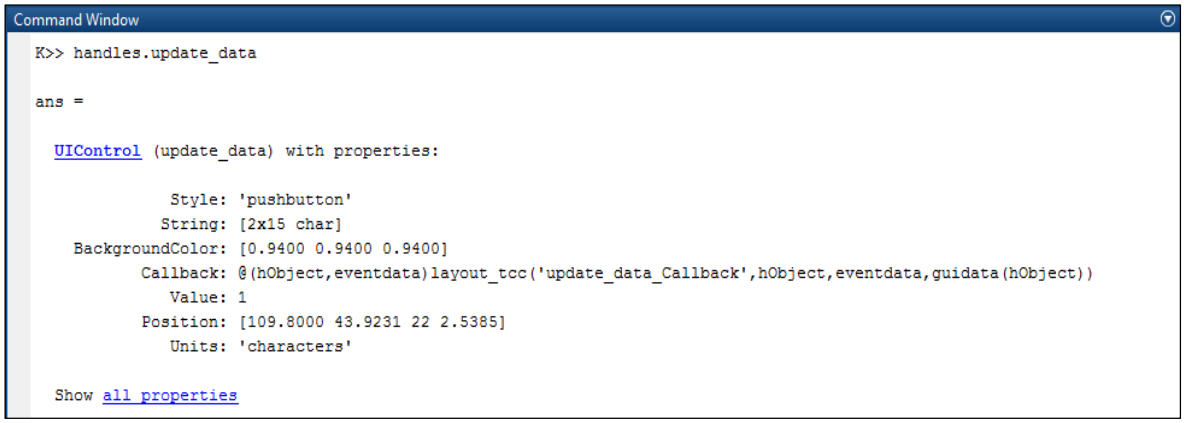


Figura 6 – Exemplo da estrutura de um elemento gráfico. Fonte: autor.

As propriedades contidas na estrutura do elemento são particulares a sua natureza. Portanto, as propriedades de um botão diferem das propriedades de um painel, ou de uma caixa de texto editável. Porém, uma propriedade que é comum a todos e que merece ser destacada, chama-se *Tag*. Cita-se essa propriedade em particular, exatamente por ser a responsável em nomear o elemento. No exemplo acima, a *Tag* do botão *Update Data* foi definida como *update_data*. Esta propriedade é de suma importância para o desenvolvedor, porque a partir dela pode-se manipular as demais, já que proporciona uma identificação do elemento. As próprias funções de interação com o usuário levam a *Tag* do elemento, como forma de individualizá-las e identificá-las.

A Fig. 7 mostra a função de criação e de *callback* de outro elemento gráfico, no caso, a caixa de texto editável que expõe a atenuação de percurso do espaço livre calculada:

```
409
410     function output_freespace_Callback(hObject, eventdata, handles)
411     % hObject    handle to output_freespace (see GCBO)
412     % eventdata  reserved - to be defined in a future version of MATLAB
413     % handles    structure with handles and user data (see GUIDATA)
414
415     % Hints: get(hObject,'String') returns contents of output_freespace as text
416     %        str2double(get(hObject,'String')) returns contents of output_freespace as a double
417
418
419     % --- Executes during object creation, after setting all properties.
420     function output_freespace_CreateFcn(hObject, eventdata, handles)
421     % hObject    handle to output_freespace (see GCBO)
422     % eventdata  reserved - to be defined in a future version of MATLAB
423     % handles    empty - handles not created until after all CreateFcns called
424
425     % Hint: edit controls usually have a white background on Windows.
426     %       See ISPC and COMPUTER.
427 -   if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
428 -       set(hObject,'BackgroundColor','white');
429 -   end
430
```

Figura 7– Funções de interação do elemento gráfico responsável pelo cálculo da atenuação de percurso do espaço livre. Fonte: autor

Fica claro ao se observar a Fig. 7, que a *Tag* deste elemento foi definida como *output_freespace*, e referencia as respectivas funções de interação.

Como todas as funções de interação recebem como parâmetro de entrada a estrutura *handles*, pode-se então manipular qualquer ferramenta gráfica e, mais especificamente, qualquer propriedade em particular, usando-se sua identificação. Para exemplificar, mostra-se uma parte do código referente à leitura dos dados fornecidos pelo usuário (Fig 8).

```
11      %%  Performance calculations
12      % receiving the inputs
13 -    B=str2num(get(handles.bandwidth,'String'));
14 -    Pt=str2num(get(handles.power_tx,'String'));
15 -    sens_rx=str2num(get(handles.sens_rx,'String'));
16 -    L_freespace=str2num(get(handles.input_freespace,'String'));
17 -    NF=str2num(get(handles.noise_figure,'String'));
18 -    Ant_Temp=str2num(get(handles.antenna_temp_rx,'String'));
19 -    L_rain=str2num(get(handles.rain_loss,'String'));
20 -    Gt=str2num(get(handles.antenna_gain_tx,'String'));
21 -    Gr=str2num(get(handles.antenna_gain_rx,'String'));
22
```

Figura 8 – Exemplo da leitura dos dados fornecidos pelo usuário a partir do *handles*. Fonte: autor

Na fig.8, acessam-se os dados das caixas de texto editável contidos na propriedade *String*. Isto compõem, então, a metodologia citada anteriormente de manipular dados a partir

das estruturas dos elementos gráficos. Este método foi usado para ler os dados de entrada, fornecer os dados de saída e implementar os gráficos.

Após explicado como foi implementado a interface gráfica, pode-se focar na explicação das funcionalidades técnicas relacionadas a telecomunicações. Primeiramente, detalha-se os gráficos do enlace.

2.4 Gráficos do Enlace

Neste programa, optou-se por estudar o terreno do enlace a partir de dois gráficos, um referente ao plano planatimétrico, e outro que representa uma visualização do enlace por satélite com zoom ótimo. Ambos os gráficos foram implementados com o elemento gráfico do tipo *Eixo* e ficam sobrepostos no programa.

Primeiramente, faz-se necessário um prelúdio teórico sobre os gráficos, para se compreender sua importância no projeto de enlace de rádio.

2.4.1 Plano Planaltimétrico

Este plano corresponde ao estudo da altitude do relevo entre os dois pontos em que se pretende fazer a comunicação em radiofrequência. A sua visualização é de extrema relevância, já que obstruções da visada direta entre as antenas de transmissão e recepção fatalmente prejudicarão a comunicação. Caracteriza-se como visada direta uma linha reta traçada entre as antenas. Garantir a visada direta não é condição suficiente para uma comunicação de qualidade, sendo um dos critérios essenciais, a obstrução da Zona de Fresnel (Haykin e Moher, 2011).

A Zona de Fresnel corresponde a elipsoides concêntricos de abertura circular que correspondem ao lugar geométrico da radiação das antenas, a qual possui este padrão de radiação devido à característica difrativa da propagação no meio.

A Fig. 9 exemplifica uma Zona de Fresnel (Haykin e Moher, 2011).

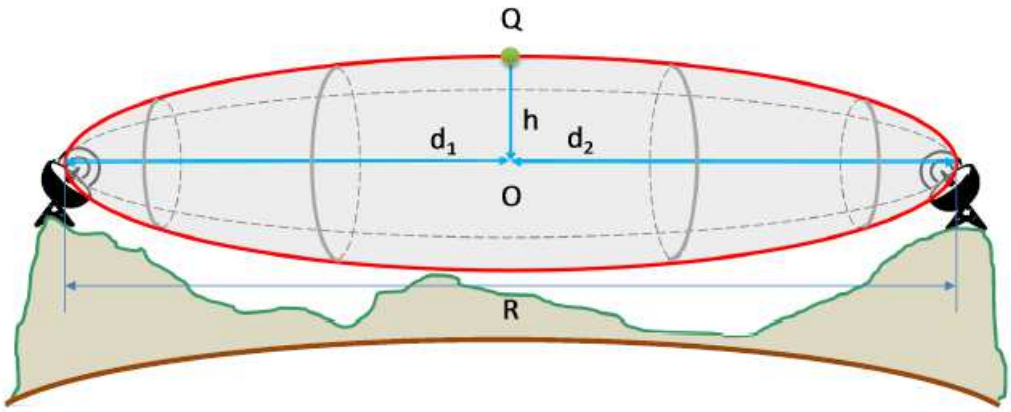


Figura 9 – Exemplo de uma Zona de Fresnel. Fonte (Haykin e Moher, 2011).

A Fig. 9 mostra o desenho da primeira Zona de Fresnel de um enlace genérico. Em teoria, existem infinitas Zonas, nas quais o raio de abertura da *n*-ésima Zona (Haykin e Moher, 2011), em qualquer ponto P é calculado por

$$h_n = \sqrt{\frac{n\lambda d_1 d_2}{d_1 + d_2}} \qquad (1)$$

Nesta equação, as distâncias d_1 e d_2 são determinadas como na figura. Já a variável λ corresponde ao comprimento de onda da portadora (Haykin e Moher, 2011). A qualidade de um enlace pode ser garantida se não houver obstrução em mais de 60% da primeira Zona de Fresnel (Haykin e Moher, 2011).

Neste contexto, torna-se imprescindível a análise do plano planaltimétrico em qualquer projeto de enlace de rádio, para assim observar possíveis obstruções e constatar a necessidade de repetidores de sinais, identificando as melhores localizações para posicioná-los. O arquivo principal responsável pela elaboração do plano planaltimétrico é o arquivo *plot_planalt.m,* o qual corresponde a uma função, e pode ser visto no Anexo 2.

Esta função recebe como parâmetros de entrada as latitudes e longitudes dos pontos que compõem o enlace, o número de amostras que se deseja ter do plano, a frequência da portadora, a altura das antenas de transmissão e recepção, e a *Tag* do elemento gráfico, para assim executar os gráficos no elemento gráfico correto. A função retorna dois vetores, sendo um correspondente à distância entre a antena transmissora e receptora, discretizada em função do número de amostras, e outro vetor de tamanho idêntico, contendo os valores

correspondentes à elevação do terreno. As altitudes do terreno são obtidas a partir da comunicação do programa com um *API Google* chamado *Google Maps Elevation API*. Este API recebe como entrada duas coordenadas geográficas e o número de amostras devolvendo, assim, as altitudes do terreno ponto a ponto, de forma equidistante, discretizadas em função do número de amostras. Portanto, se for informado o número de 500 amostras para o API, o mesmo dividirá a distância entre as coordenadas em 500 pontos equidistantes, retornando, assim, a altitude do terreno para cada um dos pontos. A versão grátis do API possui limitação de 100 requisições diárias, com uma discretização máxima de 500 pontos. Como pode-se ver no programa, tem-se um campo de entrada para o usuário escolher o número de amostras. Criou-se essa opção para que o usuário possa diminuir o número de amostras caso a conexão com a internet esteja lenta, ou o programa esteja encontrando dificuldades para processar os dados (se o *hardware* não tenha poder de processamento suficiente). Nenhum estudo de desempenho foi feito para saber o quão dramático seria em relação ao processamento, uma diminuição em amostragem, evidenciando, assim, a necessidade de um estudo para a próxima versão do *software*. De qualquer forma, recomenda-se utilizar sempre o número máximo de amostragens, pois se a distância do enlace for considerável, uma amostragem pequena pode resultar em não identificação de proeminências geográficas importantes que prejudicariam o desempenho do enlace.

A comunicação com o *API Google* foi feita com as funções de leitura de URL (*Uniform Resource Locator*) oferecidas pelo *Matlab*, mais precisamente com a função *urlread*, a qual é invocada dentro da função *plot_planalt.* Primeiramente, cria-se uma variável *string* contendo o URL padrão de acesso ao API, juntamente com os dados de entrada já descritos (coordenadas e amostragem). Esta *string* alimenta a função de leitura retornando um texto, que após certo processamento, contém os valores de altitude do terreno. A Fig. 10 mostra detalhadamente a parte do código em questão.

```
%% Google API communication
   lat(1)=lat1;
   lat(2)=lat2;
   lon(1)=lon1;
   lon(2)=lon2;
   url_p1='http://maps.googleapis.com/maps/api/elevation/xml?path=';
   url_p2='&sensor=true_or_false';
   amostras_str = num2str(amostras);
   lat_str=arrayfun(@(x) num2str(x),lat,'Un',0);
   lon_str=arrayfun(@(x) num2str(x),lon,'Un',0);
   url=[url_p1 lat_str{1} ',' lon_str{1} '|' lat_str{2} ',' lon_str{2} '&samples=' amostras_str url_p2];
    texto=urlread(url);
```

Figura 10 – Código para a comunicação com o *API Google*. Fonte: autor

Esta função invoca outros dois arquivos de extensão *m* contidos no programa:

- *Curvature.m*

  Script responsável por contabilizar a curvatura da terra. Ele contém uma sequência de comandos os quais geram uma curva com raio igual ao raio médio da terra, no sentido horário, com ponto inicial igual ao da antena de transmissão, e ponto final igual ao da antena de recepção, discretizada em função do número de amostras. Escolheu-se por utilizar um raio médio, pois identificar o raio exato da terra por coordenadas geográficas seria extremamente complicado, mas que pode ser uma ferramenta para futuras versões do *software*. Esse *script* pode ser visualizado no Anexo 3.

- *Fresnel.m*

  Função que retorna a elipse correspondente à primeira Zona de Fresnel. Apesar da Zona ser caracterizada por elipsoides, como a análise se manteve em duas dimensões, calculou-se apenas a projeção deste elipsoide em um plano, o que corresponde a uma elipse. Essa função pode ser visualizada no Anexo 4.

Após feito a importação e o processamento dos dados, a função *plot_planalt* realiza os gráficos. Mais precisamente, faz-se o gráfico da elevação do terreno em função da distância, da elevação somada à curvatura da terra em função da distância e da Zona de Fresnel. Para sobrepor estes gráficos, para assim assumir o aspecto visto no programa, faz-se necessário entender uma das propriedades do elemento gráfico *Eixo.* Esta propriedade diz ao elemento gráfico o que fazer com o próximo gráfico apontado a ele, e leva o nome de *NextPlot*.

A Fig. 11 mostra as possíveis opções que este parâmetro pode assumir.

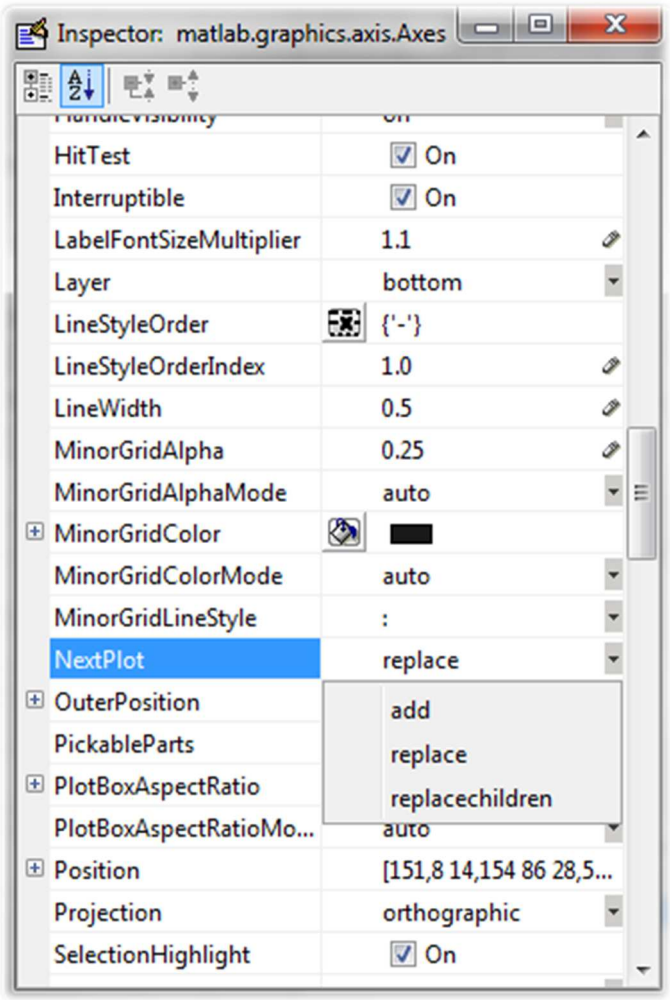


Figura 11 – Possíveis configurações para o parâmetro *NextPlot*. Fonte: autor

Ao observar a figura acima, notam-se três possíveis formas de configurar o elemento gráfico para lidar com o próximo gráfico. Quando se escolhe a opção *add*, o elemento gráfico irá sobrepor o próximo gráfico ao antigo, ou seja, os gráficos serão "somados". Já quando esta propriedade é configurada com a opção *replace*, o próximo gráfico substituirá o antigo, mas o antigo ainda continuará salvo e, a qualquer momento, é possível resgatá-lo. Com opção *replacechildren*, o elemento gráfico irá substituir o gráfico antigo pelo novo, destruindo o anterior, para não consumir memória. Pode-se observar no código do Anexo 1, mais precisamente na função de *callback* do botão *Update Data*, que esta propriedade é alterada via código diversas vezes.

A Fig. 12 mostra esta propriedade sendo trocada.

```
%% Plotting the data
legend(handles.plano_planalt,'hide'); % hide legend when a different plot is requested
set(handles.plano_planalt,'NextPlot','replacechildren'); % makes axes replace old plot
```

Figura 12 – Mudança do parâmetro responsável pelos gráficos subsequentes. Fonte: autor

2.4.2 Imagem de Satélite

Quando se quer projetar um enlace de rádio, algumas perguntas sobre a natureza do terreno impreterivelmente vêm à tona. Trata-se de uma área urbana, suburbana ou rural? Existem prédios ou construções obstruindo a visada direta? Este enlace engloba alguma propriedade privada, a qual inviabilizaria a implementação de repetidores de sinal? As antenas serão alocadas em áreas densamente populosas, em que normas mais restritivas de potência deverão ser seguidas?

Naturalmente, estas perguntas são respondidas, pelo menos parcialmente, com uma imagem de satélite do enlace e, portanto, torna-se indispensável o uso de alguma ferramenta para a visualização do terreno.

A função responsável por esta funcionalidade é a *plot_satellite.m* (Anexo 5), que recebe como entrada as coordenadas geográficas e a *Tag* do elemento gráfico *Eixo* em que será feito o gráfico. Esta função traça uma reta entre as coordenadas da seguinte forma. Usa-se uma função de interpolação de primeiro grau, chamada *polyfit*, a qual recebe as coordenadas como entrada. Com a equação da reta calculada, pode-se então traçar a reta entre os pontos. A função também chama uma outra função, a *googlemap.m* (Anexo 6). Esta última é a responsável pela comunicação com o *API Google Maps*, importando a imagem de satélite e exibindo-a com zoom ótimo. A comunicação está bem clara no algoritmo do Anexo 6 e segue a mesma lógica do gráfico do plano planaltimétrico, o qual já foi descrito anteriormente.

2.4.3 Permutação entre os gráficos

Como já antecipado anteriormente, é impossível visualizar ambos os gráficos simultaneamente, já que os elementos gráficos que carregam cada gráfico estão sobrepostos. Optou-se por esta forma de visualização como forma de economizar espaço, sendo desnecessária a criação de outras figuras e abas (já que a interface estava saturada graficamente), as quais deixariam o uso menos agradável ao usuário. Aliás, esta foi uma preocupação constante durante a síntese do *software*, sempre procurou-se aplicar soluções de

interface amigáveis para não desmotivar o seu uso, especialmente para usuários pouco familiarizados com a área de telecomunicações.

Em relação à implementação, usou-se o botão *Plot Planal/Plot Google* para realizar a permutação, naturalmente, usando sua função de *callback* (Fig. 13).

```
function mudar_plot_Callback(hObject, eventdata, handles)
% hObject    handle to mudar_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(handles.plano_planalt.Visible,'on');
    set(handles.plano_planalt,'Visible','off');
    a=get(handles.plano_planalt,'children');
    set(a,'Visible','off');
    legend(handles.plano_planalt,'hide');
    set(handles.mapa_google,'Visible','on');
    guidata(hObject,handles);
else
    set(handles.plano_planalt,'Visible','on');
    legend(handles.plano_planalt,'Terrain','Terrain considering curvature');
    a=get(handles.plano_planalt,'children');
    set(a,'Visible','on');
    set(handles.mapa_google,'Visible','off');
    guidata(hObject,handles);
end
```

Figura 13 – Função de *call-back* do botão Plot Planal/Plot Google. Fonte: autor

2.5 Coordenadas Geográficas

Há duas formas de notação muito correntes quando se quer utilizar coordenadas geográficas. A primeira consiste em uma escala que varia de 0 a 180°, aliada aos pontos cardeais, para delimitar a posição. Por exemplo, usa-se 23°N para especificar que este ponto está a 23 graus ao norte da Linha do Equador. Analogamente, a coordenada 23E indica que o ponto em questão localiza-se a 23 graus ao leste do Meridiano de Greenwich.

Já a segunda notação não utiliza os pontos cardeais, já que a variação de sua escala está no intervalo -180° a 180°. Portanto, como a escala usa valores positivos e negativos, não há a necessidade de incluir pontos cardeais para referenciar a orientação. A convenção diz que as Latitudes positivas estão a Leste do Meridiano de Greenwich e as Longitudes positivas estão ao Norte da Linha do Equador. Para o programa em questão, utilizou-se esta última notação por ser a usada nos *API Google* no qual se extraiu as informações do terreno.

Além da funcionalidade já citada, as coordenadas também são usadas para o cálculo de atenuação de chuva. Naturalmente, os níveis de precipitação pontuais dependem drasticamente da posição do enlace no globo terrestre, já que a quantidade de chuva e sua

periodicidade estão diretamente relacionadas com suas características geográficas (detalhado no capítulo *Atenuações*).

Quando se clica em *Update Data* para a realização do projeto do enlace, o programa realiza um teste para saber se as coordenadas consistem em apenas números, para se ter certeza de que não se utilizou a notação com pontos cardeais já explanada (Fig. 14).

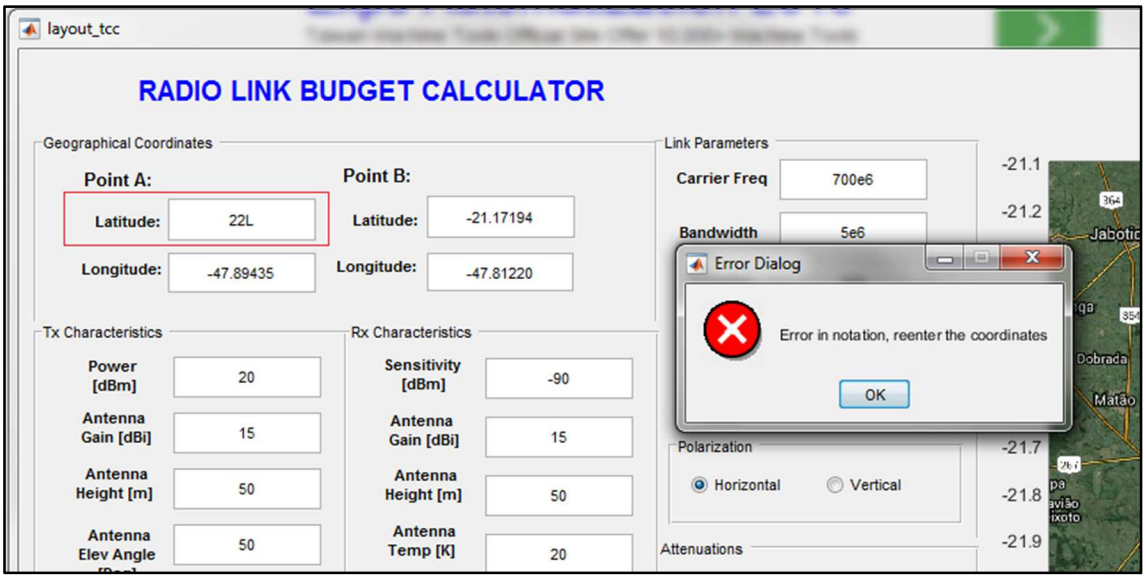


Figura 14 – Mensagem de erro quando não se utiliza a notação correta. Fonte: autor

Poder-se-ia ter implementado uma verificação adicional, relacionada aos valores limites que as coordenadas geográficas podem assumir, no caso, 180° e -180°. De qualquer forma, o levantamento das coordenadas fica a cargo do usuário, como já declarado.

2.6 Parâmetros do Enlace

Esta subseção é dedicada ao estudo dos parâmetros já antecipados no primeiro capítulo, e que estão organizados no programa sob o título de *Link Parameters*. Como estes dados caracterizam apenas entradas do usuário, pouco se tem a comentar sobre a forma em que foram implementadas no programa. Cada especificação técnica será explicada a seguir:

2.6.1 Frequência da Portadora

Primeiramente, faz-se necessário compreender qual é a função de uma onda portadora em um enlace de rádio. A onda portadora corresponde ao sinal que é transmitido e recebido pelas antenas que compreendem o enlace. Denomina-se portadora, já que sua função se resume

em portar os dados que representam a comunicação (podem estar em formato digital ou analógico). Para “inserir” os dados na onda portadora o sinal portador é modulado (BLACK *et al,* 2008, p. 8)

Pode-se considerar, por diversos motivos, a frequência da onda portadora como fator crítico para o projeto de um enlace. Primeiramente, pelo espectro de frequências, que é regulado pela ANATEL (Agência Nacional de Telecomunicações). Esta regulamentação (Resolução 625, 2013) ocorre primordialmente para se evitar interferências, respeitando assim a integridade do sinal e a privacidade da comunicação. Obviamente, as normas de regulação vão muito além da frequência da portadora, englobando potências de transmissão, alocação das antenas e outras especificações.

Além do fator legislativo, esta especificação técnica está diretamente relacionada com as antenas a serem utilizadas, e ao *hardware* em geral que se deve empregar. Por exemplo, sabe-se que as dimensões das antenas estão diretamente relacionadas com o comprimento da onda a ser transmitida (BLACK *et al,* 2008, p. 8) ou em outras palavras, a frequência da portadora.

Apesar dos motivos já apresentados, talvez o impacto mais dramático deste parâmetro esteja relacionado à atenuação que a onda sofre ao longo do seu percurso (BLACK *et al,* 2008, p. 64), sendo esta a consequência computada pelo programa (detalhes no capítulo *Atenuações*).

2.6.2 Largura de Banda

Este parâmetro está relacionado com a quantidade de dados a serem transmitidos, que em comunicação digital é dada em bits/s. Este valor é de suma importância para o projeto de um enlace, já que caracteriza um recurso valioso e escasso. Deve-se tomar cuidado para que a alocação da banda não cause interferências em canais adjacentes de comunicação, ou seja, em enlaces que utilizam portadoras com valores próximos (RAPPAPORT, 2001, p. 225). A limitação da banda também se faz importante internamente no enlace, já que está relacionada com o ruído (BLACK *et al,* 2008, p. 37) como será compreendido posteriormente neste trabalho, quando a natureza do ruído for abordada. Para se aumentar a quantidade de informação trocada no enlace, sem aumentar a banda alocada, deve-se aumentar a eficiência espectral do sistema, transmitindo conjuntos de bit codificados em símbolos (RAPPAPORT, 2001, p. 222).

2.6.3 Precipitação

A precipitação é uma forma de se quantificar a chuva, neve, gelo ou granizo, e sua unidade é expressa em milímetros por tempo decorrido. Um milímetro de chuva corresponde a um litro por metro quadrado de água sobre a superfície. No programa, decidiu-se utilizar a

medida em mm/h, já que esta forma de quantificação é usada para o cálculo da atenuação de chuva.

2.6.4 Margem de Segurança

Este parâmetro se faz necessário para considerar as demais atenuações que degradam um sinal de radiofrequência. Em decorrência da natureza heterogênea e dielétrica do ar, e do comportamento difrativo da propagação em comparação com comunicações com fio, pode-se prever que uma onda eletromagnética propagando neste meio sofrerá atenuações severas. Uma onda eletromagnética ainda sofre com reflexões ao longo do caminho, as quais podem interagir de forma destrutiva, além de estar exposta às diversas atenuações climáticas que vão além da calculada no programa (BLACK *et al,* 2008, p. 30).

Naturalmente, torna-se impossível modelar e antecipar todas as formas de atenuação com a qual um sinal desta natureza sofre ao longo do seu percurso. Portanto, para viabilizar a estimativa, calculam-se as mais severas e considera-se uma margem de segurança pessimista, ou seja, que tenha grande probabilidade de ser superior à soma das demais atenuações não contabilizadas (BLACK *et al,* 2008, p. 31).

2.6.5 Polarização

A polarização de uma onda está relacionada com o plano em que está contido a componente de campo elétrico da onda. Convencionou-se que uma onda que contém polarização vertical possui sua componente do campo elétrico no eixo *y*. Analogamente, a polarização horizontal está eixo *x*. (Haykin e Moher, 2011). A Fig. 15 mostra as duas polarizações.

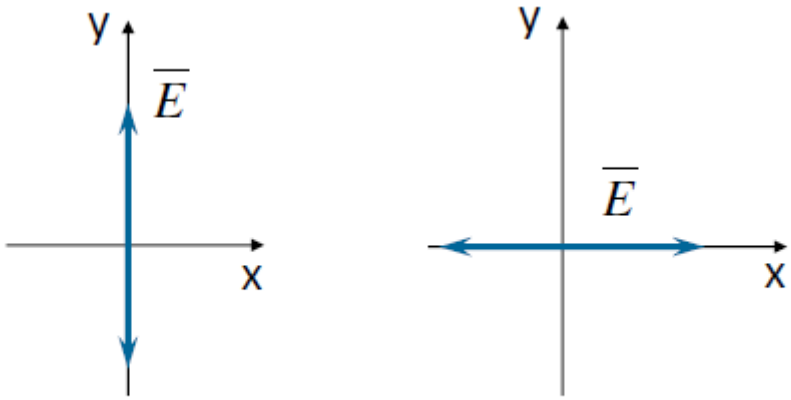


Figura 15 - Polarizações vertical e horizontal. Fonte: adaptado (Haykin e Moher, 2011)

Além das polarizações horizontal e vertical, existem também polarizações circulares e elípticas (Haykin e Moher, 2011). Porém, para o programa, considerou-se apenas as polarizações vertical e horizontal, pois estão relacionados com a atenuação de chuva calculada, a qual foi modelada apenas para estes dois tipos de polarização.

2.7 Características do Transmissor

O transmissor consiste em um conjunto de equipamentos que tem como função modular o sinal e transmití-lo. A sua configuração é particular ao método de modulação empregado e, assim, sua estrutura e funcionamento serão abordadas no capítulo *Desempenho por Modulação*. As características do transmissor que devem ser informadas pelo usuário serão detalhadas a seguir:

2.7.1 Potência

Esta é uma das características mais críticas do ponto de vista financeiro do projeto de um enlace. Obviamente, transmissões de altas potências necessitam de *hardwares* mais robustos, que são mais caros, e consomem muita energia, impactando no custo de operação e manutenção do enlace (BLACK *et al,* 2008, p. 63). Além do fator econômico, há também a legislação (Resolução 625, 2013), como já antecipado nesta mesmo subseção. A emissão de micro-ondas é regulamentada por normas baseadas em estudos sobre efeitos biológicos de exposição (BLACK *et al,* 2008, p. 63).

No programa, quantificou-se essa grandeza em unidade dBm. Esta unidade logarítmica considera a referência 1 miliwatt. A fórmula para se transformar uma potência em watts para dBm é

$$P_{dBm} = 10log_{10}\left(\frac{P[W]}{1mW}\right) \quad (2)$$

2.7.2 Ganho da Antena

Para se entender este parâmetro característico de antenas, necessita-se primeiramente compreender o conceito de uma antena isotrópica, a qual por definição, irradia ondas eletromagnéticas com igual intensidade independentemente da direção (BLACK *et al,* 2008, p. 20). Esta antena ideal serve apenas como padrão de referência para se estabelecer a quantificação do ganho. Portanto, quando uma antena apresenta ganho de 15 dBi, entende-se que sua irradiação é 15 vezes superior, em uma escala logarítmica, que a de uma antena isotrópica, em uma direção em particular (BLACK *et al,* 2008, p. 23). Compreende-se então as chamadas antenas direcionais, pois são caracterizadas por apresentar ganho elevado em uma

direção específica, apresentando um diagrama de radiação bem característico, onde quase todo o seu ganho se concentra em um lóbulo estreito (BLACK *et al,* 2008, p. 27). O ganho de uma antena sem perdas é dado por

$$G\ [dBi] = \frac{4\pi}{\lambda^2} A_e \qquad (3)$$

Em que λ corresponde ao comprimento de onda e A_e à área efetiva de abertura. Este último parâmetro está relacionado com a eficiência de radiação (BLACK *et al,* 2008, p. 27).

2.7.3 Altura da Antena

O impacto deste parâmetro é visto no gráfico do plano planaltimétrico, mais precisamente no desenho da primeira Zona de Fresnel. Como já explicado anteriormente, a não obstrução desta Zona é um critério essencial para uma comunicação de qualidade. Portanto, torna-se necessário instalar antenas em altitudes elevadas, posicionadas em proeminências geográficas para minimizar as possíveis obstruções. Em contrapartida, instalações feitas em altitudes elevadas também podem dificultar a manutenção, tanto na logística operacional quanto em custo. Portanto, pode-se experimentar diferentes altitudes no programa para se encontrar uma posição viável para o usuário.

2.7.4 Ângulo de Elevação

Este ângulo corresponde ao ponto de maior diretividade da antena, em relação a um plano horizontal. Este parâmetro tem influência no cálculo da atenuação de chuva. A Fig. 16 exemplifica um ângulo de elevação (CHARLESWORTH, [s. d.], p. 1).

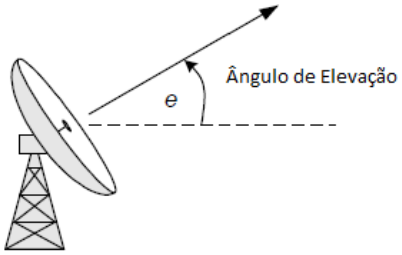


Figura 16 – Ângulo de elevação da antena. Fonte: (CHARLESWORTH, [s. d.], p. 1).

2.8 Características do Receptor

O receptor consiste em um conjunto de equipamentos os quais têm a função de receber a onda portadora e demodulá-la, reconstituindo a informação transmitida. As mesmas considerações em relação ao transmissor são válidas para o receptor.

2.8.1 Sensibilidade do Receptor

Esta figura de mérito tem como função delimitar um valor mínimo de potência de sinal para que o receptor consiga captá-lo e processá-lo (BLACK *et al,* 2008, p. 30). O programa acusa com um sinal de alerta caso a potência calculada do sinal que chega no receptor seja inferior à sensibilidade informada. O código responsável por esse teste está na Fig. 17.

```
% checking if received power is lower than receiver's sensitivity
if ~isempty(get(handles.sens_rx,'String'))
    if sens_rx>Pr_dBm
        warndlg('Received Power is lower than receivers sensibility','Warning');
        movegui(gcf,'center');
    end
end
```

Figura 17 – Código responsável por alertar o usuário que a potência que chega no receptor é inferior à informada. Fonte: autor

Na caixa de texto abaixo dos gráficos, onde se faz considerações de cunho técnico sobre o enlace, pode-se encontrar o valor da potência recebida pelo receptor. Fica a cargo do usuário, com o auxílio do programa, encontrar uma potência suficiente para que o desempenho do sistema seja satisfatório.

2.8.2 Temperatura da Antena

Esta unidade representa a temperatura hipotética de um resistor, que produziria uma potência equivalente à potência de ruído produzido pela antena (BLACK *et al,* 2008, p. 39).

Antenas e equipamentos de radiofrequência em geral, estão sujeitos a diferentes fenômenos geradores de ruído das mais diversas naturezas, os quais degradam o sinal recebido (BLACK *et al,* 2008, p. 34).

Portanto, este parâmetro tem como função contabilizar todos os ruídos presentes na antena, como se fossem de natureza térmica. A fórmula para calcular o ruído térmico é

$$Potência\ do\ Ruído\ térmico\ [W] = kTB \qquad (4)$$

Em que k corresponde à constante de Boltzmann, T corresponde à temperatura do equipamento, e B à largura de banda. Como já foi dito, a largura de banda é um fator crucial na geração de ruído (BLACK *et al,* 2008, p. 46).

É possível combinar diferentes temperaturas de ruído para se calcular uma temperatura equivalente, contabilizando todo o ruído de um sistema. A fórmula para este cálculo é

$$T\,[K] = T_1 + \frac{T_2 - 1}{G_1} + \frac{T_3 - 1}{G_1 G_2} + \cdots + \frac{T_n - 1}{G_1 G_2 \ldots G_{n-1}} \qquad (5)$$

Onde G_n é o ganho disponível de potência do *n*-ésimo amplificador. Nota-se que os índices das variáveis estão relacionados à ordem que os equipamentos estão conectados. Para um estudo mais detalhado recomenda-se a referência (BLACK *et al,* 2008, p. 54).

2.8.3 Figura de Ruído

O fator de ruído de um equipamento mostra quanto um equipamento em particular influencia na relação entre a potência do sinal, e a potência do ruído. O fator de ruído é definido como

$$F = \frac{SNR_{entrada}}{SNR_{saída}} \qquad (6)$$

Em que SNR $(Signal\ to\ Noise\ Ratio)$ representa a razão sinal-ruído (BLACK *et al,* 2008, p. 51). Por se tratar de uma razão, seu uso corrente é feito em escala logarítmica. Quando utilizado nesta escala, este parâmetro é conhecido por figura de ruído,

$$NF = 10log_{10}(F) \qquad (7)$$

É importante ressaltar que o fator de ruído nunca pode ser inferior a um (ou a figura de ruído inferior a zero, em escala logarítmica) já que todo equipamento adiciona ruído. Como as fontes são de naturezas diversas, modela-se a soma dos ruídos como se fosse exclusivamente térmica (BLACK *et al,* 2008, p. 45).

Como fator de ruído, e temperatura equivalente, são ambas figuras de mérito relacionadas ao mesmo fenômeno, pode-se estabelecer uma equivalência entre elas por meio de

$$Tequivalente\ [K] = T_o(F - 1) \tag{8}$$

Em que T_o corresponde à uma temperatura de referência do ambiente, geralmente 290 K (BLACK *et al,* 2008, p. 54). No programa, representou-se o ruído da antena a partir de uma temperatura de ruído equivalente, e o ruído do receptor por sua figura de ruído. Escolheu-se por estas duas terminologias diferentes, já que antenas geralmente tem seu ruído representado desta forma, enquanto que equipamentos eletrônicos terrestres, em geral, têm seu ruído caraterizado segunda esta outra notação (BLACK *et al,* 2008, p. 51).

Fatores de ruído de vários equipamentos também podem ser associados para se obter um fator de ruído total (BLACK *et al,* 2008, p. 55), de um sistema em particular. Analogamente à temperatura equivalente de ruído, a equação da associação de múltiplos fatores é

$$F = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \cdots + \frac{F_n - 1}{G_1 G_2 \ldots G_{n-1}} \tag{9}$$

Portanto, pode-se resumir brevemente como foi modelado todo o ruído presente na associação do receptor com a antena, no programa em questão. Primeiramente, transformou-se a figura de ruído do receptor em temperatura equivalente e calculou-se a temperatura equivalente de todo o sistema. Desta forma, o programa consegue contabilizar a relação sinal-ruído que chega no demodulador, a qual tem papel fundamental na análise de desempenho do sistema, como ficará claro no capítulo *Desempenho por Modulação.* Os fragmentos de código responsáveis por estes cálculos são mostrados a seguir.

```
% calculating power and noise
Teq = Ant_Temp + (290*(power(10,NF/10) - 1) - 1)/power(10,Gr/10); %equivalente temp noise
```

Figura 18 – Código responsável por contabilizar o ruído do receptor. Fonte: autor.

2.9 Atenuações

Como já descrito anteriormente, existem inúmeras formas de atenuação que uma onda eletromagnética sofre ao propagar em um meio (BLACK *et al,* 2008, p. 34). No programa, optou-

se por calcular as duas atenuações consideradas mais severas, em relação à perspectiva do desenvolvedor (nenhum estudo foi feito para comprovar se essas atenuações são de fato as mais significantes), deixando as demais serem representadas pela margem de segurança. Como na descrição dos gráficos, faz-se necessário um prelúdio teórico em relação à ambas as atenuações para justificar ao leitor, a necessidade de implementá-las.

2.9.1 Atenuação do Espaço Livre

Esta atenuação está atrelada ao modelo de propagação admitido neste contexto, o qual leva o nome de Modelo de Propagação Livre, um enlace em que não há obstruções.

Este modelo de propagação é aplicado em distâncias suficientes para se considerar propagação de campo distante. A literatura faz distinção entre regiões próximas e distantes das antenas (BLACK *et al,* 2008, p. 22) e neste programa são considerados os campos distantes, A propagação de ondas eletromagnéticas, sob estas circunstâncias, é modelada pela *Fórmula de Friis*:

$$\frac{P_r}{P_t} = G_t G_r \left(\frac{\lambda}{4\pi R}\right)^2 \tag{10}$$

Em que P_r corresponde à potência de recepção, P_t à potência de transmissão, G_t ao ganho da antena transmissora, G_r ao ganho da antena receptora, λ ao comprimento da onda portadora e R à distância do enlace (BLACK *et al,* 2008, p. 56). A atenuação de percurso é dada por

$$Atenuação\ [dB] = \left(\frac{4\pi R}{\lambda}\right)^2 = \left(\frac{4\pi R f}{c}\right)^2 \tag{11}$$

Em que f é a frequência da portadora, e c corresponde à velocidade a luz (BLACK *et al,* 2008, p. 32).

Como existe uma função em *Matlab* para o cálculo desta atenuação, sua implementação foi imediata.

O código responsável pelo cálculo desta atenuação se Fig. 19:

```
% Free Space Loss
lambda = physconst('LightSpeed')/freq;
fspace_loss = fspl(1000*handles.dist(end),lambda);
aux = num2str(fspace_loss);
set(handles.output_freespace,'String',aux);
clear aux;
```

Figura 19 – Código que calcula a atenuação de espaço livre. Fonte: autor.

2.9.2 Atenuação de Chuva

Um sinal eletromagnético é atenuado em condições chuvosas devido principalmente ao espalhamento que sofre quando percorre as gotas, e pela absorção do sinal pelas moléculas de água. Esta absorção é negligenciável para neve e gelo, em que as moléculas não possuem a mesma liberdade de movimentação (NELSON, 2000, p. 1).

Esta atenuação se torna significativa a medida que a onda portadora atinge comprimento de ondas próximos ao tamanho das gotas de chuva. Como uma gota de água possui um tamanho próximo de 1,5 mm, frequências próximas de 40 GHz (que corresponde à um comprimento de onda de aproximadamente 7,5 mm) já apresentam atenuações de chuva consideráveis (NELSON, 2000, p. 1). Em condições climáticas mais severas, com níveis de precipitação mais elevados, as gotas de água apresentam tamanho maiores, caracterizando atenuações mais significativas para frequências de operação menores (NELSON, 2000, p. 1).

Esta atenuação é comumente calculada por

$$L_r[dB] = kR^{\alpha}D = \gamma D \tag{12}$$

Em que L_r corresponde à atenuação da chuva em dB, R à precipitação em mm/h, D à um percurso equivalente em km, e k e α correspondem a parâmetros empíricos os quais são função da frequência de operação e da polarização utilizada (NELSON, 2000, p. 3).

A forma como estes parâmetros são calculados ou estimados dependem das diferentes metodologias para calculá-los (CHARLESWORTH, [s. d.], p. 2). Neste trabalho, para implementar o algoritmo responsável por calcular a atenuação de chuva, usou-se do método *ITU-R Recommendations P.* 838. Os passos que mostram esta metodologia sendo aplicada se encontram no Apêndice B, e o código responsável no Anexo 8.

2.10 Desempenho por Modulação

Modulação é uma prática que tem como intuito adicionar informações a uma onda portadora. Isto é possível alterando as características da onda, como fase, frequência e amplitude (BLACK *et al,* 2008, p. 226).

A modulação analógica consiste em alterar a frequência (FM) e amplitude (AM, conforme mostra a Fig. 20 (RAPPAPORT, 2001, cap 5).

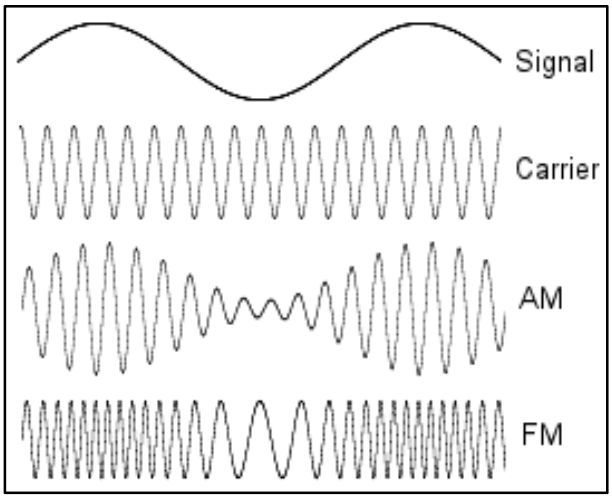


Figura 20– Exemplo de modulações em amplitude e frequência.
Fonte: adaptado de (Haykin e Moher, 2011).

Como se vê na Fig. 20, as variações de amplitude e frequência são proporcionais à excursão do sinal modulante. Para se obter o sinal original, após a transmissão, o sinal é demodulado.

Atualmente, a modulação digital é mais utilizada (BLACK *et al,* 2008, p. 343).

A Fig. 21 exemplifica a modulação digital em amplitude, fase e frequência.

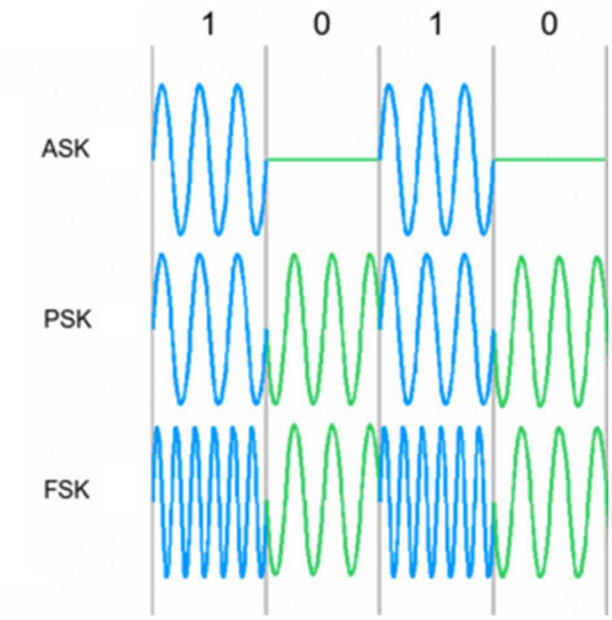


Figura 21 – Exemplo de modulação digital. Fonte: adaptada de (Moher e Haykin, 2011).

Na modulação digital, a informação a ser transmitida é representada por bits. Para comparar o desempenho dos sistemas que utilizam modulação digital, define-se o parâmetro BER (*Bit-Error-Rate*), como uma razão média entre bits errados e transmitidos. A expressão “bits errados” caracteriza, por exemplo, a situação em que se transmitiu um bit “1”, e no processo de demodulação, o sistema interpretou-o erroneamente como “0”. Este erro ocorre devido à interação do sinal com o rúido, e, portanto, diminui à medida que se aumenta a razão sinal-ruído presente no demodulador. Os valores de BER aceitáveis estão relacionados com a aplicação e dependem do quão fidedigna e precisa deve ser a informação reconstituída no processo de demodulação (BLACK *et al,* 2008, p. 62). A forma com que o rúido caracteriza a fórmula do BER, para as diferentes modulações empregadas no programa, se encontra detalhada no Apêndice B.

2.10.1 Implementação das modulações

A região do programa em que se deve escolher o método de modulação, e onde se visualiza o respectivo BER, se encontram na Fig. 22.

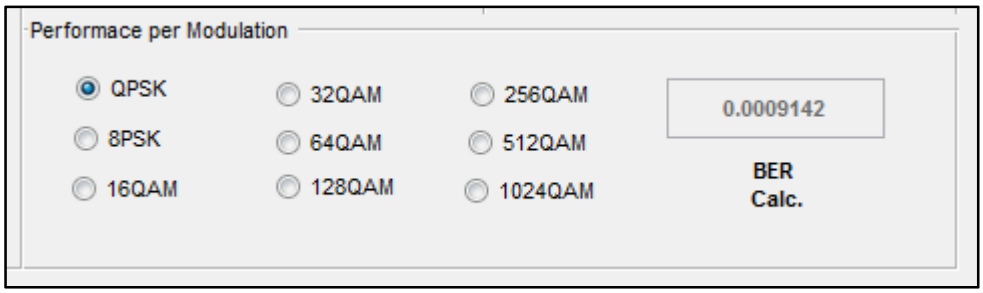


Figura 22 – Área dedicada ao programa para estudar as modulações e seu desempenho. Fonte: autor

A implementação do cálculo do desempenho do sistema para cada modulação foi facilitada pelo uso da função *berawgn* do *Matlab*, a qual calcula o BER para diversos métodos de modulação considerando o ruído como aditivo, branco e gaussiano. Portanto, a implementação apenas usou-se de uma lógica simples para identificar qual método está selecionado pelo usuário para calcular corretamente o BER. Vale lembrar que a lógica se baseou na associação dos elementos gráficos “Grupo de botões” e “Botões de rádio”, como já explicado. O código responsável por esses cálculos se encontra integralmente dentro do arquivo *tech_parameters.m* (Anexo 7).

Para se calcular o BER do sistema, deve-se saber de antemão a relação sinal-ruído final que chega no demodulador. Para o cálculo deste parâmetro, primeiramente, o programa calcula a potência do sinal que chega no receptor sabendo-se dos ganhos das antenas, potência de transmissão e as atenuações ao longo do percurso, segundo a fórmula de Friis. Em seguida, contabiliza-se a degradação do sinal no receptor, a partir da temperatura da antena e da figura de ruído do equipamento, as quais são englobadas em uma temperatura equivalente. Resumindo, o programa utiliza a seguinte equação (BLACK *et al,* 2008, p. 52):

$$SNR = \frac{Pr}{kTB} \tag{13}$$

Em que o termo Pr corresponde à potência recebida. Já o termo do denominador corresponde ao ruído térmico equivalente do sistema. Com este valor em mãos, usa-se as fórmulas de BER para calcular o desempenho do sistema.

2.11 Manipulações de dados

Nesta subseção, faz-se o compromisso de explicar o funcionamento dos dois botões referentes à manipulação dos dados do programa, os quais se encontram agrupados como mostra a Fig. 23:

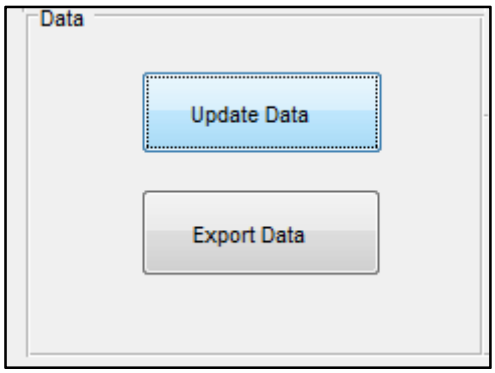


Figura 23 – Botões para manipular dados no programa. Fonte: autor

O botão *Update Data,* como já explicado, é o responsável por todos os cálculos do programa, já que os códigos referentes a todos os cálculos se encontram na função de *callback* deste elemento gráfico. Esta função pode ser visualizada no Anexo 1.

Primeiramente, a função em questão torna visível o botão *Export Data* já que em sua configuração inicial, o botão está invisível. Isto foi implementado para que o usuário não clique por engano no botão *Export Data* sem se ter calculado o enlace, o que resultaria em erros no programa. Em seguida, todos os dados de entrada são capturados em variáveis, testa-se a notação das coordenadas geográficas, e chama-se os arquivos referentes ao estudo do terreno, os quais já foram abordados. O botão então chama os arquivos *tech_parameters.m,* o qual calcula as atenuações, a relação sinal-ruído final no demodulador e sua respectivo desempenho, e *Link_considerations.m,* o qual configura estes valores calculados para entrar na caixa de texto logo abaixo dos gráficos, fornecendo informações úteis sobre o enlace.

Em relação ao botão *Update Data,* sua função de *call-back* pode ser vista a seguir:

```
function export_data_Callback(hObject, eventdata, handles)
% hObject    handle to export_data (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(handles.plano_planalt.Visible,'off');
    mudar_plot_Callback(hObject,eventdata,handles);
end
run('export_data.m');
```

Figura 24 – Função de *call-back* do botão de exportar os dados. Fonte: autor.

Para melhor organizar o código, optou-se por separar o código referente à exportação em outro arquivo, o qual pode ser visto no Anexo 9. Neste arquivo, o código procura os elementos do tipo “caixa de texto editável” armazenando seus valores para entrar no documento a ser exportado. Em seguida, chama-se o arquivo responsável pela exportação, o qual utiliza-se da toolbox *MATLAB Report Generator*, como antecipado no começo desse trabalho. Este arquivo possui extensão *rpt* e consiste em uma linguagem própria, na qual é desenvolvida dentro do aplicativo em *Matlab*, referente à esta toolbox. A imagem a seguir mostra o código em desenvolvimento no aplicativo em questão:

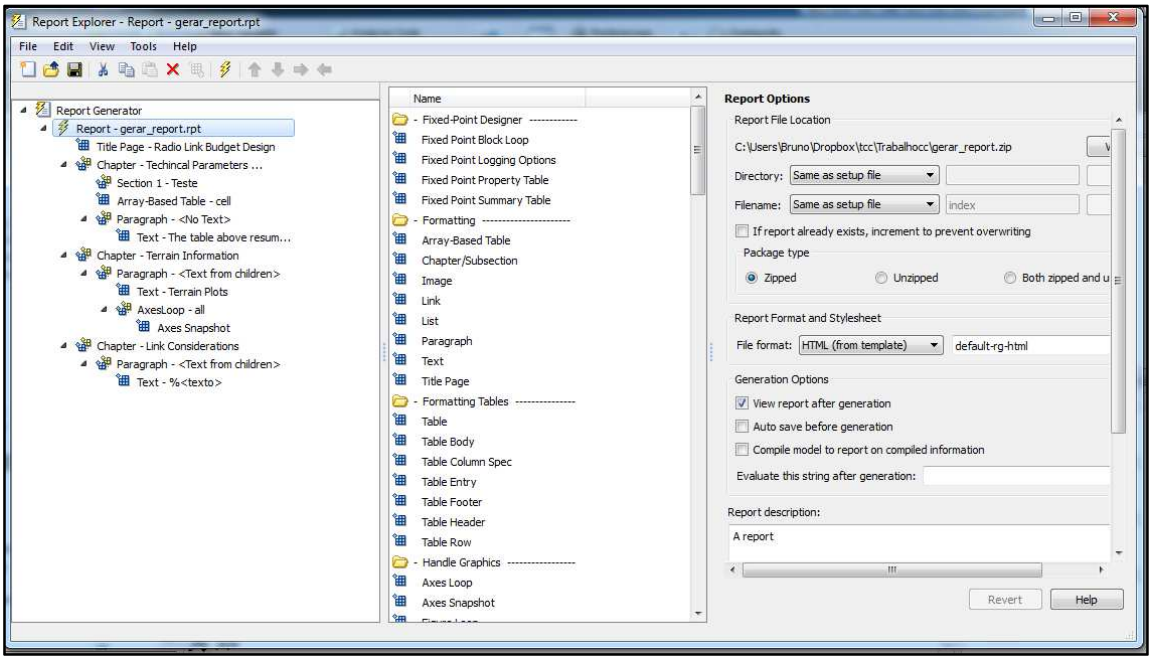


Figura 25 – Aplicativo para gerar relatórios do MATLAB com o código do programa em questão.
Fonte: autor

O aplicativo funciona da seguinte forma. Escolhe-se os elementos referentes à texto, como parágrafo, título, texto, tabelas e vários outros, os quais interagem com certa hierarquia, da coluna central da figura acima. Já a coluna à esquerda mostra o código desenvolvido para o programa usando-se destes elementos. Pode-se notar que alguns elementos estão contidos dentro de outros, o que mostra a disposição hierárquica dos elementos. Para exemplificar, como pode-se prever intuitivamente, um texto está sempre contido em um parágrafo, que por sua vez está contido em um título. Esses elementos são alimentados por variáveis criadas em *Matlab*, caracterizando assim a forma como os dados se comunicam com o arquivo de geração de relatório.

Como já sugere o código, o relatório gerado pelo programa está intitulado como *Radio Link Budget Design,* e está divido em três capítulos, os quais levam o nome de *Technical Parameters, Terrain Information e Link Considerations.* Cada um será explicado sucintamente a seguir:

- Technical Parameters (Parâmetros Técnicos)

  Nele está contido uma tabela a qual mostra todos os parâmetros do enlace, desde os que o usuário forneceu até os calculados pelo programa.

- Terrain Information (Informações do Terreno)

  Contêm os dois gráficos referentes ao terreno do enlace realizados pelo programa.

- Link Considerations (Considerações do Enlace)

  Este capítulo simplesmente exporta as considerações que se localizam embaixo dos gráficos.

  Um exemplo de exportação dos dados se encontra no Anexo 10.

3. VALIDAÇÃO DA FERRAMENTA

Este capítulo tem como função validar os cálculos contidos no programa, comparando-os com os programas *PathCheck* e *RF Haversine Lite*. Cada funcionalidade será testada a seguir:

3.1 Estudo do Terreno

Primeiramente, faz-se uma simples validação da visualização de satélite do enlace, entrando no programa, coordenadas geográficas conhecidas e observando se o programa retorna os locais corretos. Usando as coordenadas geográficas das cidades de São Carlos e Ribeirão Preto, ambas localizadas no Estado de São Paulo.

Tabela 1 – Coordenadas geográficas das cidades de São Carlos e Ribeirão Preto.

São Carlos	**Ribeirão Preto**
-22.00709° de latitude	-21.17194° de latitude
-47.89435° de longitude	-47.81220° de longitude

Ao entrar com estas coordenadas geográficas, o programa retorna o seguinte gráfico:

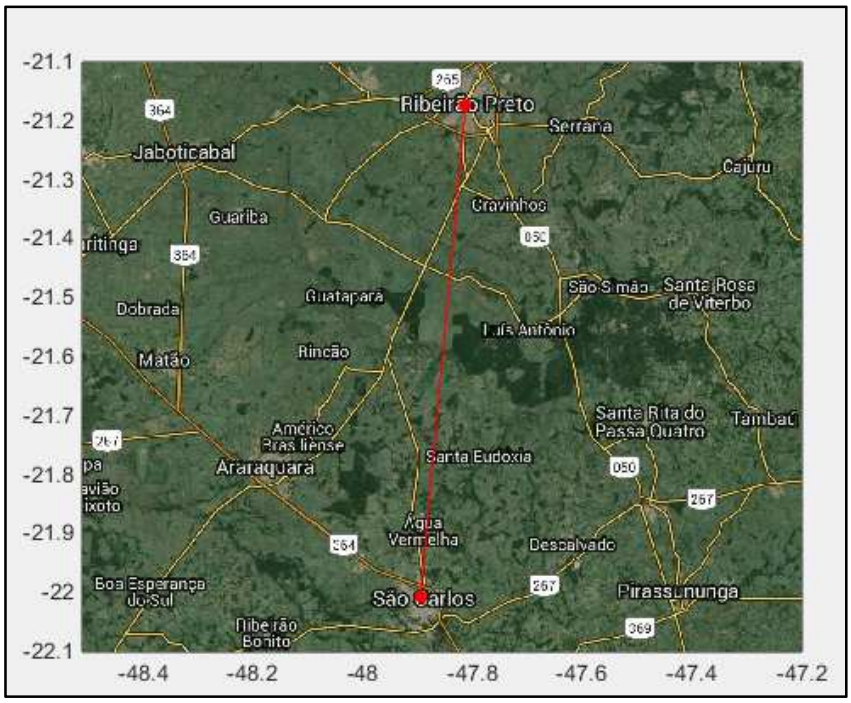


Figura 26 – Visualização do enlace por satélite. Fonte: autor

Claramente o programa está retornando os locais corretos segundo as coordenadas informadas pelo usuário. Quanto ao erro desta funcionalidade, pode-se encontrar mais informações pesquisando a documentação do API utilizado.

Já o plano planialtimétrico retornado pelo *software* pode ser visualizado na Fig. 27. Para os planos planatimétricos, assumiu-se 50 metros de altura para as antenas.

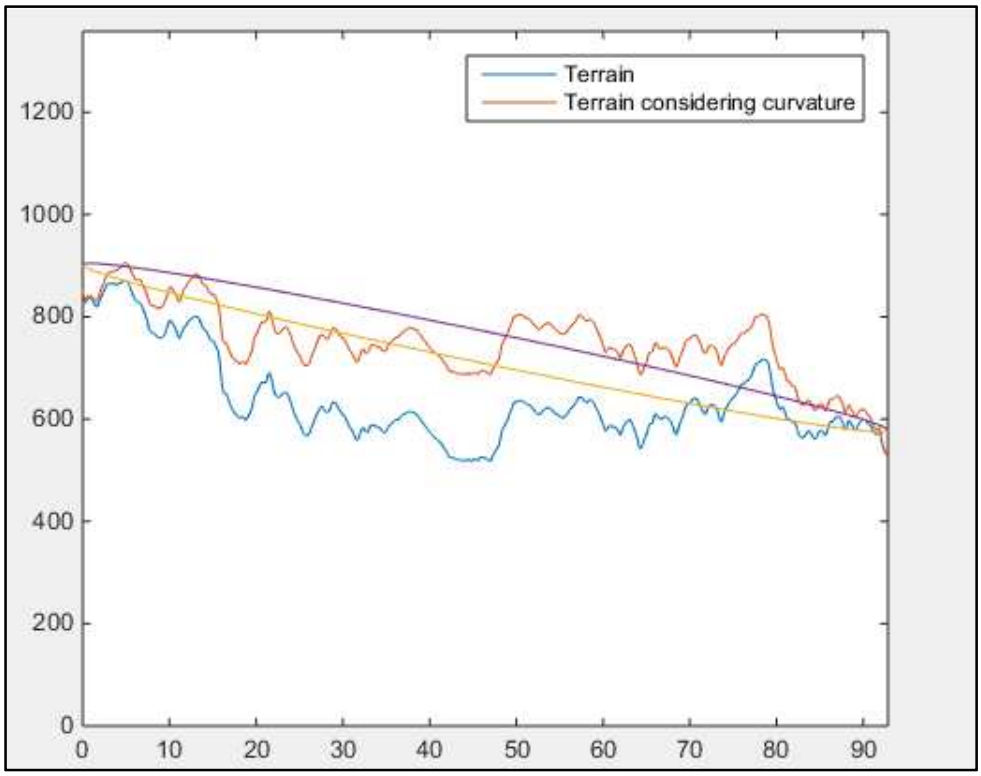


Figura 27 – Visualização do plano planaltimétrico do terreno entra as cidades de São Carlos/SP e Ribeirão Preto/SP. A elipse da figura se refere à primeira Zona de Fresnel. Fonte: autor

Os planos planaltimétricos retornados pelos programas referência se encontram nas figuras a seguir:

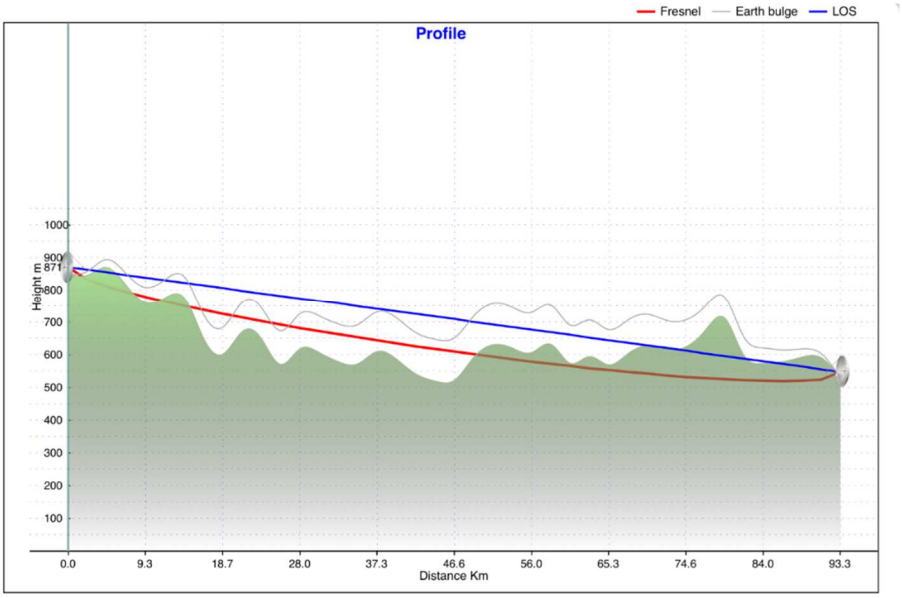


Figura 28 – Plano Planaltimétrico do programa *Haversine* referente ao enlace compreendido entre as cidades de São Carlos e Ribeirão Preto. Fonte: *Haversine*

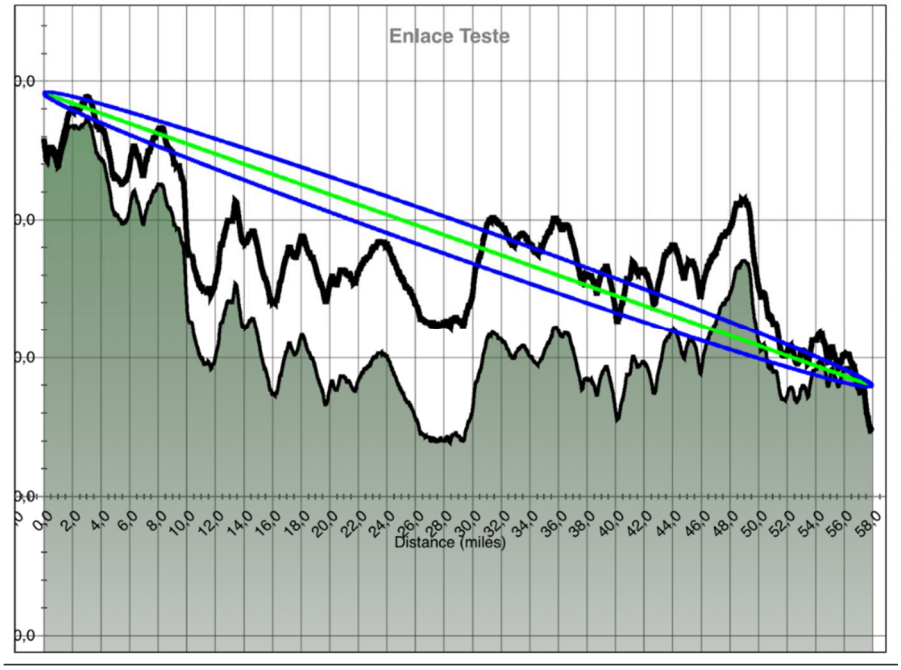


Figura 29 - O plano planaltimétrico fornecido pelo programa *PathCheck.* Fonte: *PathCheck*

Ao se comparar os planos, pode-se concluir que todos acusam perfis de terreno extremamente semelhantes, coincidindo na localização de proeminências e depressões. Talvez a única diferença evidente entre eles, se encontra em sua resolução. O programa *Haversine*

demonstra um perfil suave, sem reentrâncias e recortes, já o programa *PathCheck* e o desenvolvido neste trabalho, apresentam as imperfeições do relevo, as quais coincidem absolutamente. A tabela a seguir mostra a comparação dos demais parâmetros relacionados ao terreno:

Tabela 2 – Comparação dos parâmetros relacionados ao terreno.

	Haversine	***PathCheck***	**Programa a ser validado**
Altitude do Transmissor [m]	851,27	Não informado	851,46
Altitude do Receptor [m]	528,20	Não informado	528,20
Distância do Enlace [km]	93,28	93,25	92,86

A partir desta tabela, é possível identificar uma ligeira diferença entre a distância do programa a ser validado, com os outros. Ainda sim, a diferença apresentada é percentualmente insignificante.

Portanto, a partir dos dados apresentados, pode-se afirmar que as funcionalidades relativas ao estudo do terreno, estão validadas.

3.2 Atenuações

A tabela seguir compara os valores de atenuação de espaço livre, do enlace em questão. Usou-se uma frequência de portadora de 8 GHz para o cálculo:

Tabela 3 – Comparação das atenuações de percurso do enlace em estudo.

	Haversine	***PathCheck***	**Programa a ser validado**
Atenuação de percurso [dBm]	149,9	149,9	149,8664

Ao se observar a tabela, fica-se claro que a ligeira diferença entre as distâncias evidenciada na Tabela 2, não impactou em diferenças no cálculo da atenuação de espaço livre.

Já em relação à atenuação de chuva, não foi possível fazer a validação a partir dos programas de referência, já que os mesmos não apresentam esta atenuação discriminada. Assim, para testar esta funcionalidade, usou-se o exemplo da referência (CHARLESWORTH, [s. d.]), no qual exemplifica-se o cálculo de uma atenuação de chuva.

Neste exemplo, usa-se de um enlace localizado na Dinamarca, operando com uma frequência de portadora igual a 8 GHz. O enlace ainda considera uma polarização horizontal, um ângulo de elevação de antena igual 25 graus, uma estimativa de precipitação de 6mm/hr e uma

altura de antena de 0 metros. Para estas condições, a atenuação de chuva calculado pelo exemplo é de 0,22 dB.

Primeiramente, para se considerar este enlace no programa em questão, necessita-se saber as coordenadas geográficas. O exemplo delimita a latitude de 52° para a Dinamarca, mas nada informa sobre a longitude assumida. Portanto, decidiu-se pelo valor de 10° de longitude, para viabilizar a validação. Estas coordenadas são referentes à posição geográfica do transmissor. Sabendo-se que o cálculo desta atenuação, na forma que foi implementada no programa, leva em consideração a posição apenas do transmissor, escolheu-se uma posição geográfica arbitrária para o receptor, no caso, 52.1° de latitude e 10.1° de longitude.

O resultado do teste pode ser visto no Fig. 30.

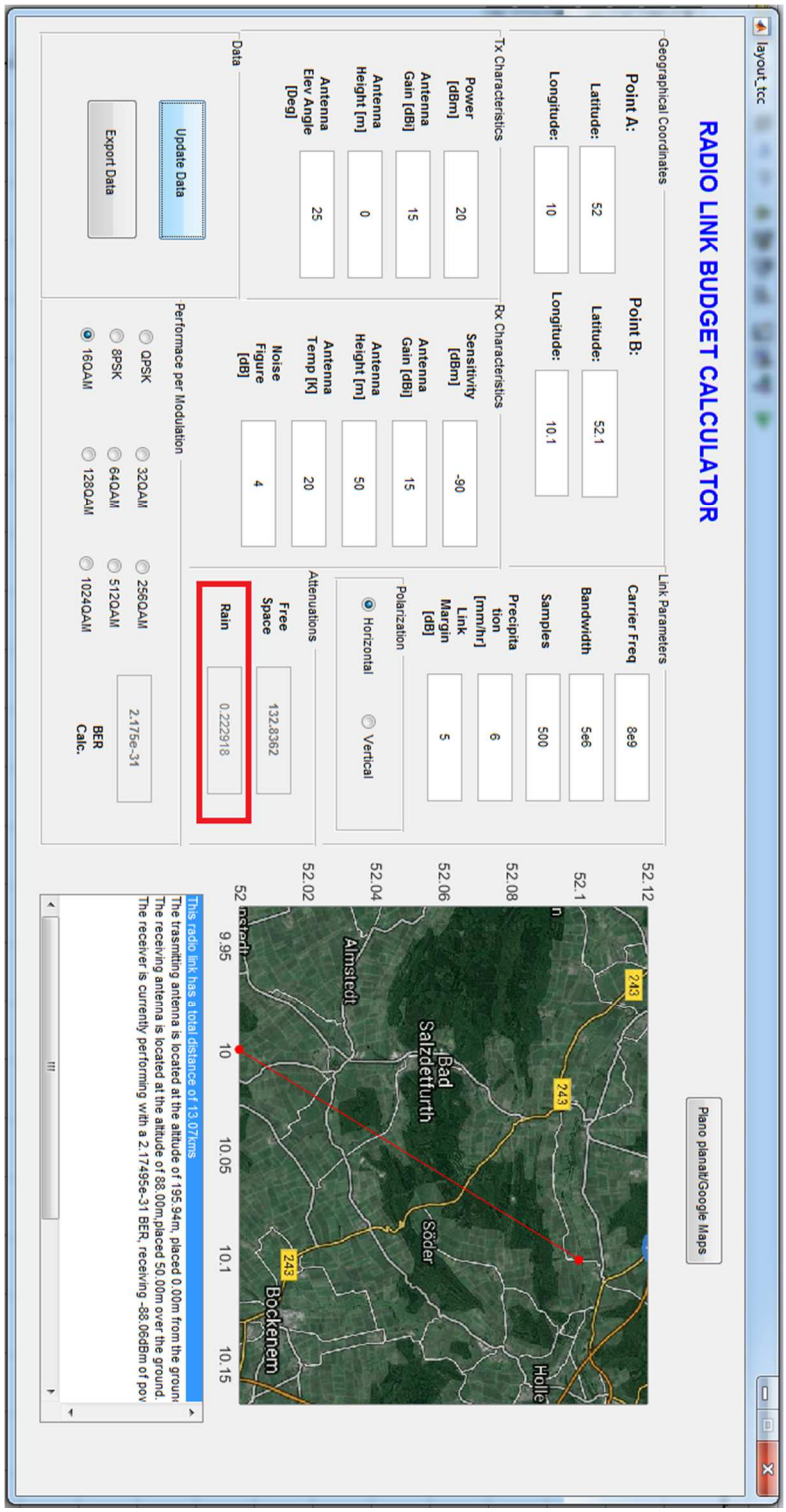


Figura 30 – Sáida do programa com as entradas de atenuação de chuva do exemplo referência.

A atenuação de chuva está evidenciada por um retângulo vermelho. Como pode-se ver, a atenuação calculada, após arredondada para a segunda casa decimal, apresenta valor igual à 0,22 dB, como calculada no exemplo. Nota-se que não se preocupou com os demais parâmetros para este enlace em particular, já que seu cálculo teve como intuito apenas a validação da atenuação de chuva.

Como o cálculo deste parâmetro apresenta certa complexidade, precisar-se-ia realizar mais testes, com valores de latitudes diferentes, para se afirmar que esta funcionalidade está validada. Infelizmente, não se encontrou mais cálculos como o da referência, de outras partes do mundo, para a realização da validação integral. Porém, a partir do teste realizado, pode-se considerar que esta implementação tem grande chance de estar integralmente correta, ou seja, retornando a atenuação de chuva correta para qualquer latitude.

3.3 Desempenho por Modulação

Faz-se desnecessário validar as funções de cálculo de BER, já que as mesmas foram desenvolvidas pela *MathWorks,* e assim, pode-se assumir com segurança de que as mesmas já passaram por um processo de validação. O que seria necessário validar, está relacionado à forma que se calculou a relação sinal-ruído, parâmetro de entrada destas funções. Infelizmente não se encontrou uma forma de comparar os valores calculados, já que o cálculo da relação sinal-ruído final é muito particular à forma que se modelou e agrupou as figuras de mérito relacionadas à ruído. Em outras palavras, não se conseguiu elaborar um teste em que se pudesse entrar com dados no programa, para validar o desempenho. Os programas de referência também nada informam sobre este parâmetro.

4. RESULTADOS E DISCUSSÕES

Neste capítulo, detalhem-se os problemas, soluções, alternativas e dúvidas, que se fizeram presentes ao longo do desenvolvimento do programa.

Primeiramente, pode-se destacar como um dos grandes desafios da síntese deste programa, a concepção e realização da interface gráfica. Gastou-se muito tempo tentando-se entender como se organizam as estruturas gráficas do *Matlab,* e como manipulá-las. Em especial, as realizações dos gráficos foram talvez as implementações que mais tomaram tempo no desenvolvimento, já que a relação do elemento tipo *Eixo* com seus respectivos gráficos, não se encontra bem documentada. Ainda em relação à este elemento, o mesmo apresenta algumas características que dificultaram as implementações executadas para este programa. Uma delas, para qual não se encontrou uma verdadeira solução, está relacionada a função *plot quando* referenciada à um elemento do tipo *Eixo* em especial. Em alguns casos, após chamada a função, suas propriedades características eram *resetadas* para valores *default* do programa, perdendo sua identificação, e impossibilitando assim, que se conseguisse manipular o elemento. Portanto, ao longo do programa, teve-se que reconfigurar as *Tags* desses elementos, de forma a não deixá-las perderem a identificação. Esta prática pode ser vista no Anexo 1, mais especificamente no *call-back* do botão *Update Data.* Toda vez que este botão é acionado, faz-se necessário realizar os comandos demonstrados na Fig. 29. Infelizmente não foi possível encontrar uma explicação para esta perda de identificação, e assim, usou-se desta alternativa para viabilizar a realização dos gráficos.

```
set(handles.mapa_google,'Tag','mapa_google');
set(handles.plano_planalt,'Tag','plano_planalt');
```

Figura 31– Parte do código em que se teve que reconfigurar o parâmetro de identificação dos elementos gráficos do tipo eixo. Fonte: autor.

Em relação ao cálculo de atenuação de chuva, mais especificamente à estimativa da precipitação, necessita-se expor as seguintes considerações. Inicialmente pensou-se em estimar este parâmetro de forma automatizada, como já dito anteriormente. Pesquisou-se exaustivamente diversos APIs de bancos de dados climáticos, e algumas alternativas foram encontradas. Alguns bancos oferecem a opção de se conhecer as precipitações anuais ou mensais de muitos anos atrás, o que a princípio solucionaria o problema, pois com pouca

manipulação matemática, poderia-se identificar uma precipitação pessimista em relação ao espaço amostral fornecido pelo banco. Em contrapartida, os dados fornecidos eram sempre valores médios de regiões extensas, como países ou continentes, perdendo assim qualquer significado em relação ao cálculo pontual da atenuação. Já nos demais bancos, em que se pode extrair informações geograficamente pontuais, ou seja, a partir das coordenadas geográficas, limita-se a consulta da precipitação em dias (versão grátis), não fornecendo assim espaço amostral suficiente para se tomar qualquer conclusão estocástica. Infelizmente, esta funcionalidade parou de funcionar, já que o identificador utilizado na construção do URL expirou. Tem-se que renovar a chave de acesso no site do API.

A alternativa foi deixar à cargo do usuário encontrar uma precipitação pessimista para se calcular a atenuação. Felizmente, a referência (CHARLESWORTH, [s. d.]) demonstra uma metodologia para estimá-lo. Uma outra alternativa foi implementada, mas que merece as seguintes considerações e ressalvas. Esta alternativa é acionada quando o usuário não fornece nenhuma precipitação ao programa. O software então se comunica com um API fornecido pela *OpenWheaterMap*, para extrair as precipitações pontuais das coordenadas geográficas referentes à antena transmissora, dos últimos 16 dias no período de 3 horas que antecedem o horário do pedido. O algoritmo responsável procura então a pior precipitação deste período, e o utiliza para calcular a atenuação de chuva. O código que faz a comunicação com o API e realiza esta lógica se encontra no Anexo 8.

Desnecessário dizer que esta atenuação não caracteriza de forma alguma uma estimativa segura de atenuação, mas ainda sim decidiu-se por implementar esta alternativa pelo seguinte motivo. Este banco de dados climáticos, como muitos outros, fornece não gratuitamente, o acesso à dados históricos de precipitações pontuais. Portanto, caso o usuário seja assinante de algum banco, com pequenas alterações no código, pode-se implementar a automatização da precipitação.

Para o gerenciamento dos arquivos que compõem o programa, usou-se da ferramenta *Simulink Project,* como antecipado na introdução. Infelizmente não se conseguiu usufruir de suas funcionalidades, pois ao longo do desenvolvimento do *software,* ocorreram problemas com a pasta de repositório do arquivo. Em consequência, não se conseguiu realizar um controle de versões como se havia pensando inicialmente, já que o *Matlab* não conseguia acessar os arquivos contidos na pasta de repositório. Descobriu-se que o problema com a pasta estava relacionado com a forma que fora nomeada, já que apresentava caracteres proibidos não reconhecidos pelo *Matlab.*

Decidiu-se por dividir o código do programa, em diversos arquivos, como forma de melhor visualizar a função de cada um, e poder assim entender os processos por trás dos cálculos. Os códigos se encontram integralmente comentados, passo a passo. Todos os arquivos se encontram nos anexos.

Como prometido na introdução, exportou-se o programa para torná-lo independente de seu ambiente de desenvolvimento, e assim funcionar em um sistema operacional *Windows*. A geração do executável, a partir do aplicativo *MATLAB Compiler,* ocorreu normalmente, sem erros, e o resultado pode ser visto na figura a seguir

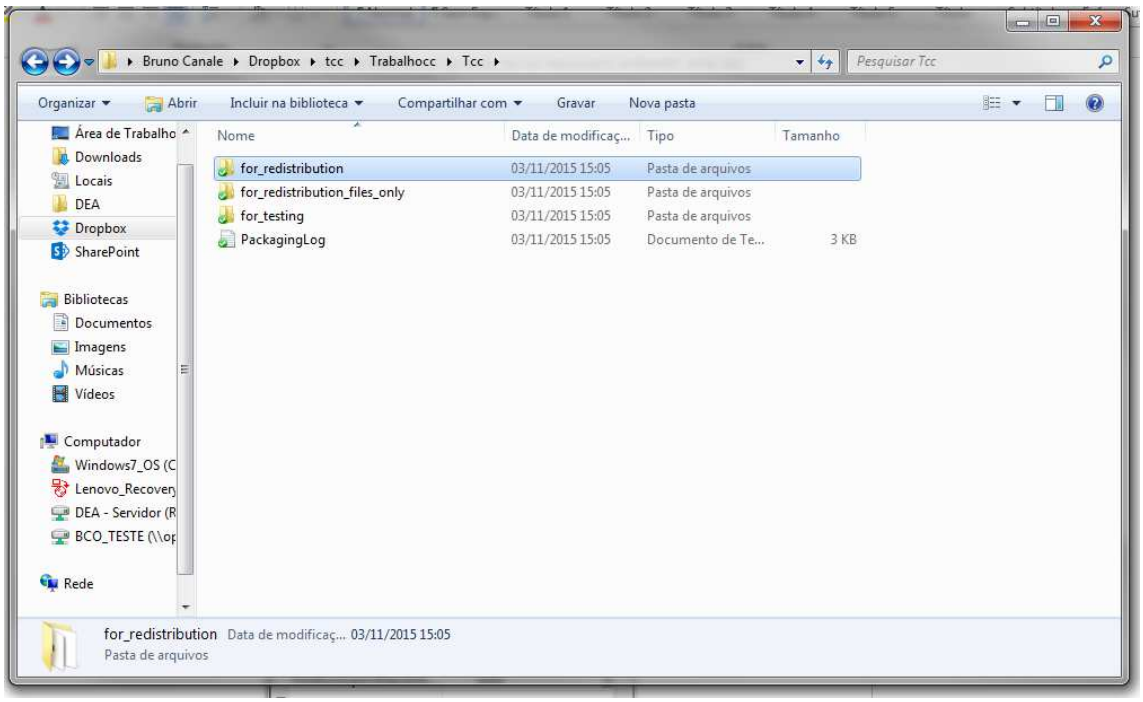


Figura 32 – Pasta que contêm os arquivos do programa final. Fonte: autor

Pode-se ver que o aplicativo retorna uma pasta contendo outras três, na qual uma contêm o instalador, uma segunda que contêm as imagens do programa (logo, imagem de fundo e outras) e um executável, e a terceira com arquivos auxiliares que podem ser configurados no aplicativo, como o arquivo *readme,* o qual é bem comum em diversos softwares (tem como função auxiliar na instalação).

A Fig. 33 mostra o programa sendo instalado.

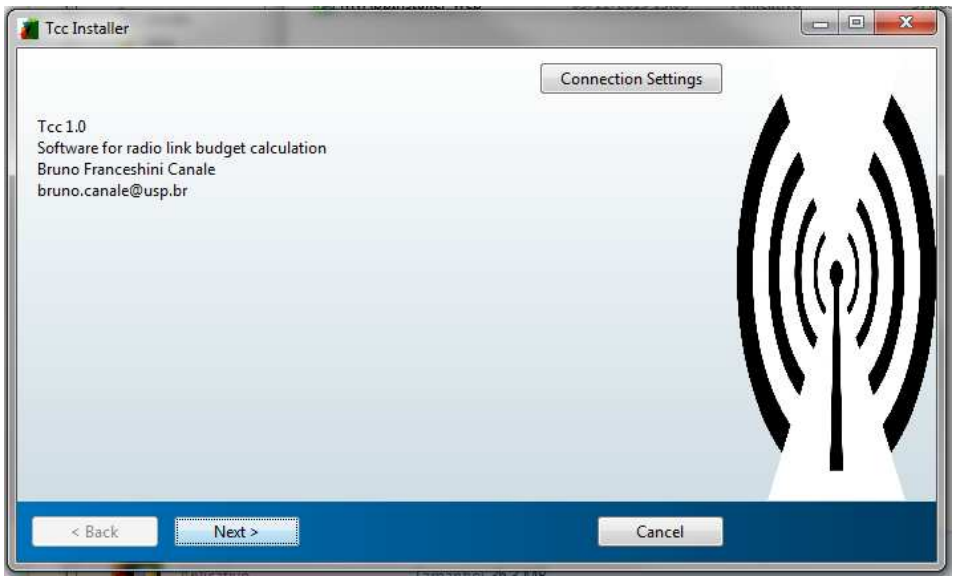


Figura 33 – Instalação do programa. Fonte: autor

A instalação não apresentou erro. A Fig 34 mostra o programa devidamente instalado.

Figura 34 – Programa devidamente instalado. Fonte: autor.

Apesar de, aparentemente, o processo de exportação da ferramenta tenha ocorrido sem erros, já que o aplicativo não acusou nenhum problema, ou aviso de qualquer natureza, o

programa não funcionou corretamente. A sua inicialização ocorre sem problemas, pode-se mudar os valores nas caixas de texto editável, mas ao apertar o botão *Update Data,* o programa trava. A Fig. 35 mostra exatamente o momento descrito.

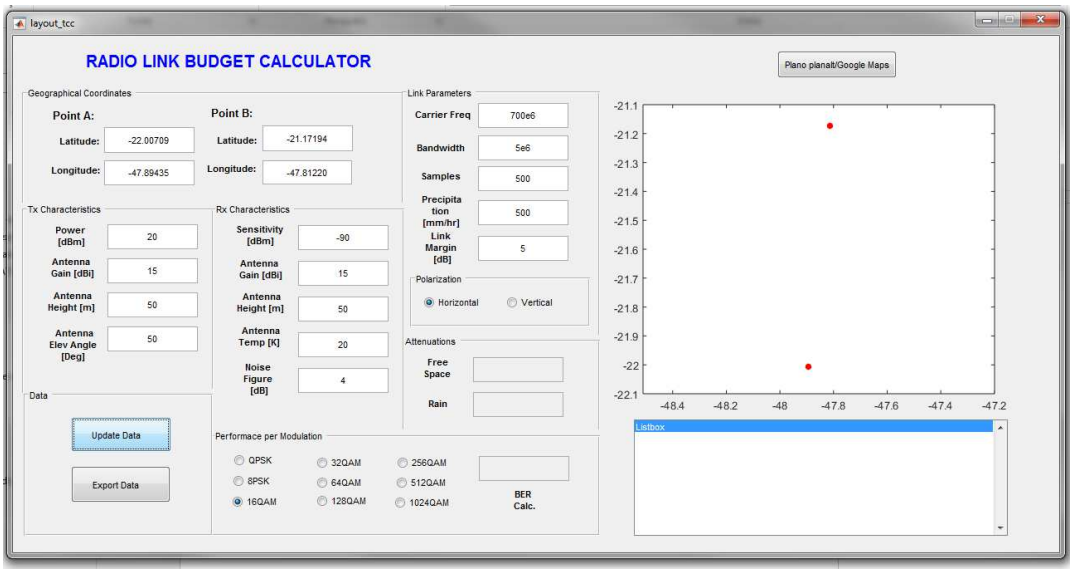


Figura 35 – Programa ao parar de funcionar. Fonte: autor.

Pode-se observar que o programa para de funcionar, quando tenta executar os comandos relacionados à geração da imagem de satélite do enlace, em que o código realiza um requerimento pelo API Google, para assim receber a imagem. Esta observação se prova pelo fato de que o *software* desenha os pontos comunicantes e para logo em seguida. Ao analisar o Anexo 5, pode-se ver que o comando seguinte à realização dos pontos, é responsável pela geração da imagem de satélite. Após acionado o suporte da *Mathworks*, foi possível isolar o fator responsável. O problema se encontrava no momento em que a imagem de satélite era importada do API. Alguma rotina de segurança do *Windows* impossibilitava esta ação, causando um erro no programa. A solução encontrada consiste em executar o programa no modo Administrador, permitindo uma certa liberdade ao programa em relação às rotinas de segurança do *Windows.* Ao configurar que o programa seja executado sempre no modo Administrador, este erro não mais aparece, e o programa funciona corretamente.

Para facilitar o uso do *software,* criou-se um *Help* que pode ser acessado no canto superior esquerdo do programa. Neste *Help,* faz-se uma breve descrição do programa, seguida das instruções de uso, e das definições das saídas do programa. Nas definições, é possível

encontrar *links* que explicam os fenômenos calculados, a modelagem matemática utilizada, e em alguns casos, até exemplos.

Infelizmente, ao longo da escrita deste trabalho, criou-se o hábito de identificar a antena do ponto A como transmissora, e a antena do ponto B como receptora. Naturalmente, já que se trata de uma comunicação de duas vias (*duplex*), ambas as antenas transmitem e recebem, caracterizando assim a notação utilizada como incorreta.

5. CONCLUSÃO

Pode-se afirmar que o trabalho atingiu suas metas e objetivos. Conseguiu-se criar um *software* de telecomunicações, o qual foi submetida à processos de validação, para auxiliar o *projeto* de enlaces de rádio, apresentando uma modelagem razoavelmente semelhante à realidade, gratuito e com código aberto. O seu compartilhamento será feito por uma plataforma que a *MathWorks* oferece para a troca de códigos, conhecida como *Link Exchange.*

Quanto às metas pedagógicas, o programa mostrou sua capacidade de fornecer uma visualização de como os parâmetros se relacionam em um enlace, proporcionando a absorção de conceitos básicos e valioso de telecomunicações. O programa também instiga a curiosidade pelo ponto de vista de projeto de *softwares*, já que apresenta uma interface interessante, cálculos estatísticos e determinísticos, comunicações com API e outras funcionalidades curiosas.

6. REFERÊNCIAS BIBLIOGRÁFICAS

RAPPAPORT T. S. (2001). ***Wireless Communications: Principles and Practice.*** 2ed. NJ, EUA: *Prentice Hall.*

CHARLESWORTH P. ([s. d.]). ***Rain Fade Calculations:*** *Satellite Communications lectures.* Notas de Aula. New Port: *University of Wales.*

BLACK B. A. *et al* (2008). ***Introduction to Wireless Systems.*** 1ed. USA: *Prentice Hall*

NELSON R. A. (2000). ***Rain How it Affects the Communications Link:*** Course Notes. Maryland: *Applied Technology Institute.*

MOHER M. e HAYKIN S. (2011). **Sistemas de Comunicação.** 5ed. *Bookman.*

TILAL M. (2014). ***Digital Modulation:*** Course Notes. Attock: *Institute of Information Technology.*

MathWorks. Disponível em: < http://www.mathworks.com/ >. Acesso em: 02 de novembro de 2015.

Matlab Central Link Exchange. Disponível em: < http://www.mathworks.com/matlabcentral/linkexchange/?term=maps+googl+zor >. Acesso em: 06 de outubro de 2015.

ANATEL. Legislação. *Resolução nº 625, de 11 de novembro de 2013.* Disponível em < http://www.anatel.gov.br/legislacao/resolucoes/2013/644-resolucao-625 >. Acessado em: 04 de novembro de 2015.

APÊNDICE A – Cálculo da atenuação de chuva segundo o método *ITU-R Recommendations P. 838*

Primeiramente, para calcular a atenuação, deve-se calcular o parâmetro D, o qual pode ser melhor explicado a partir da visualização da Fig. 1.

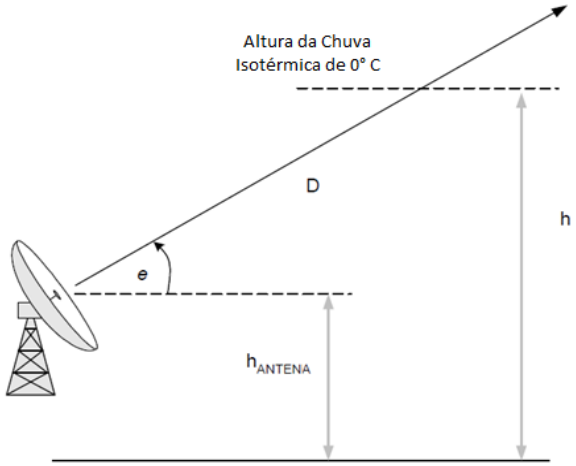


Figura 1 – Definição do percurso equivalente D. Fonte (CHARLESWORTH, [s. d.], p. 1)

Como se pode observar, este percurso está relacionado com a altura da antena e do seu ângulo de elevação, o qual é representado pelo termo e, além do parâmetro h, o qual se refere à altura entre o chão e uma faixa de 0°C na atmosfera. Naturalmente, para o cálculo do termo D, se faz a suposição de que a atmosfera é composta de faixas isotérmicas, e que toda chuva é originada em uma delas, precisamente na que possui temperatura correspondente ao ponto de congelamento da água (CHARLESWORTH, [s. d.], p. 1).

O cálculo do h é possível sabendo-se a latitude e longitude do local em particular, seguindo a lógica da tabela abaixo (CHARLESWORTH, [s. d.], p. 2).

Tabela 1 – Diferentes fórmulas para o cálculo do h em função dos valores de latitude e longitude. Fonte (CHARLESWORTH, [s. d.], p. 2)

Latitude ϕ	***h* (altura da chuva)**	**Região**
$\varphi > 23N$	$5 - 0{,}075(\varphi - 23)$	Hemisfério Norte (exceto América do Norte e Europa)
$0 \leq \varphi \leq 23N$	5	Hemisfério Norte (exceto América do Norte e Europa)
	$3.2 - 0{,}075(\varphi - 35)$	América do Norte e Europa à oeste da longitude 60E
$21S \leq \varphi \leq 0$	5	Hemisfério Sul
$71S \leq \varphi < 21S$	$5 + 0{,}1(\varphi + 21)$	Hemisfério Sul
$\varphi < 71S$		Hemisfério Sul

Seguramente teve-se que realizar a mudança de notação de coordenadas geográficas, para implementar esta lógica. Considerou-se que a América do Norte está compreendida para longitudes inferiores -60°, e que a Europa Ocidental se encontra na faixa de longitudes maiores que -20° e menores que 60°. O código responsável para calcular o h se encontra na Fig. 2.

```
%% Calculating h_rain:
if ((latA <= 0) && (latA > -21))
    h_rain = 5;
elseif (latA <= -21)
    h_rain = 5 + 0.1*(abs(latA) + 21);
elseif ((latA > 0) && (lonA<-60 || ((lonA > -20) && (lonA < 60))))
    h_rain = 3.2 - 0.075*(abs(latA) - 35);
elseif (((latA > 0) && (latA <= 23)) && (((lonA >= -60) &&...
        (lonA <= -20)) || (lonA >= 60)))
    h_rain = 5;
elseif ((latA > 23) && (((lonA >= -60) && (lonA <= -20)) || (lonA >= 60)))
    h_rain = 5 - 0.075*(abs(latA) - 23);
end
```

Figura 2 – Algoritmo para o cálculo do h. Fonte: autor.

Assim, sabendo-se a altura da antena, o parâmetro h (altura da faixa isotérmica de congelamento da água) pode-se calcular o parâmetro D com uma simples relação trigonométrica (CHARLESWORTH, [s. d.], p. 2):

$$D = \frac{(h - h_{ANTENA})}{Sen(e)} \quad (1)$$

Em seguida, deve-se saber a precipitação mais pessimista da região em particular, segundo um grau de confiança. A partir da divisão do mapa terrestre em zonas, as quais discriminam diferentes taxas de precipitação, pode-se observar a precipitação correta para o cálculo segundo a tabela 2 (CHARLESWORTH, [s. d.], p. 2).

Tabela 2 – Precipitação por zonas em função do grau de confiança.

Fonte (CHARLESWORTH, [s. d.], p. 2).

Porcentagem do tempo que R é excedido	Zona														
	A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q
1,0	< 0,1	0,5	0,7	2,1	0,6	1,7	3	2	8	1,5	2	4	5	12	24
0,3	0,8	2	2,8	4,5	2,4	4,5	7	4	13	4,2	7	11	15	34	49
0,1	2	3	5	8	6	8	12	10	20	12	15	22	35	65	72
0,03	5	6	9	13	12	15	20	18	28	23	33	40	65	105	96
0,01	8	12	15	19	22	28	30	32	35	42	60	63	95	145	115
0,003	14	21	26	29	41	54	45	55	45	70	105	95	140	200	142
0,001	22	32	42	42	70	78	65	83	55	100	150	120	180	250	170

Já α e β são parâmetros empíricos que podem ser encontrados segundo a tabela 3 (CHARLESWORTH, [s. d.], Apêndice 2).

Tabela 3 – Parâmetros empíricos para o cálculo da atenuação de chuva em função da frequência de operação e da polarização utilizada. Fonte (CHARLESWORTH, [s. d.], Apêndice 2)

Frequência em GHz	**Polarização Horizontal**		**Polarização Vertical**	
	k	α	k	α
1	0,0000387	0,912	0,0000352	0,880
2	0,000154	0,963	0,000138	0,923
4	0,000650	1,121	0,000591	1,075
6	0,00175	1,308	0,00155	1,265
7	0,00301	1,332	0,00265	1,312
8	0,00454	1,327	0,00395	1,310
10	0,0101	1,276	0,00887	1,264
12	0,0188	1,217	0,0168	1,200
15	0,0367	1,154	0,0335	1,128
20	0,0751	1,099	0,0601	1,065
25	0,124	1,061	0,113	1,030
30	0,187	1,021	0,167	1,000
35	0,263	0,979	0,233	0,963
40	0,350	0,939	0,310	0,929

Esta tabela foi adicionada ao programa, mais especificamente na função de iniciação da figura de interface. Assim, toda vez que o programa abre, a tabela é carregada automaticamente. É interessante esclarecer que as latitudes e longitudes usadas para o cálculo da atenuação são da antena transmissora.

APÊNDICE B – Cálculo do BER para os diferentes métodos de modulação, a partir da modelagem do ruído.

O ruído pode ser modelado como sendo branco e gaussiano, já que apresenta potência constante ao longo de todo o seu espectro, como a luz branca, a qual possui emissões uniformes em todo o espectro visível, e possui distribuição normal (BLACK *et al,* 2008, p. 36). A função de densidade de probabilidade do ruído branco gaussiano tem o seguinte aspecto:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \quad (1)$$

Em que μ representa a média e σ^2 a variância. Assume-se que o ruído possui média igual a zero, e variância:

$$\sigma^2 = \frac{N_0}{2} \quad (2)$$

Onde N_0 corresponde à densidade espectral de ruído, a qual tem sua unidade expressa em W/Hz.

Para entender como o ruído e as modulações empregadas se relacionam matematicamente, primeiramente mostra-se um método simples de modulação digital, o BPSK.

1.1 BPSK

BPSK, ou *Binary Phase-Shift Keying,* corresponde à um método de modulação em fase, para representar bits. Mais especificamente, representa-se um sinal digital modulando a onda portadora com dois valores de fase distintos, cada qual representando os dois níveis lógicos do sistema binário. Portanto, a partir do valor da fase recebida no receptor, o mesmo consegue distinguir o bit transmitido. Geralmente, usa-se uma forma de visualização dos possíveis valores de fase e amplitude que a portadora pode assumir, correspondentes com o sinal digital a ser transmitido. Esta metodologia leva o nome de Diagrama de Constelação. Para entendê-lo integralmente, faz-se necessário o uso de algumas ferramentas da álgebra linear.

A imagem a seguir mostra um exemplo das alterações de fase e os bits correspondentes, caracterizando o diagrama de constelação de uma modulação BPSK (RAPPAPORT, 2001, p. 238). Vale afirmar que a realização deste diagrama é uma prática corrente e comum a todos os métodos de modulação.

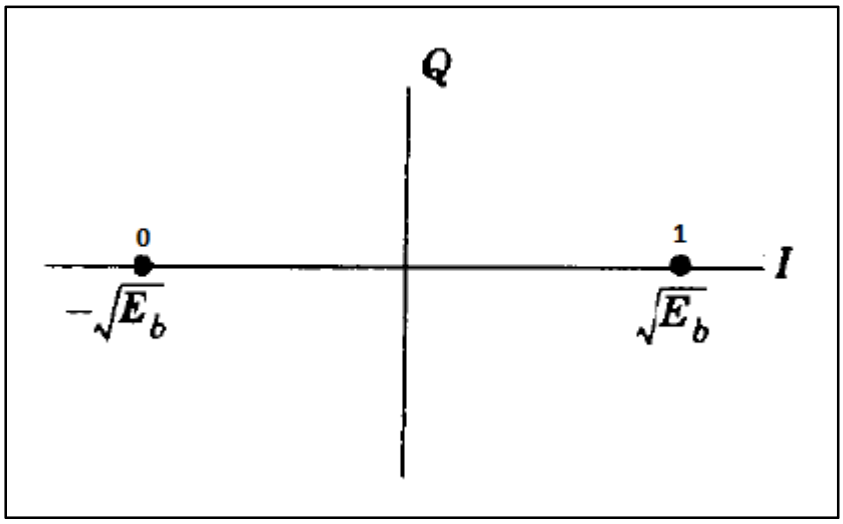


Figura 1 – Diagrama de constelação de uma modulação BPSK.

Fonte: (RAPPAPORT, 2001, p. 237)

A Fig. 1 contém a representação de dois sinais em um plano referencial, cada qual representando um bit a ser transmitido. Para a criação de um diagrama, precisa-se de dois eixos ortogonais, para que se possa representar os sinais que a portadora pode assumir. Obviamente, no caso desta constelação, apenas uma dimensão já seria suficiente para representar ambos os sinais, porém, como será visto nas modulações aplicadas no programa, para transmissões de grupos de bits, duas dimensões serão necessárias. Para dois sinais representarem uma base, os mesmos devem ser ortogonais entre si, o que se traduz em matemática na seguinte equação (RAPPAPORT, 2001, p. 239):

$$\int_{-\infty}^{\infty} \emptyset_i(t)\emptyset_j(t)dt = 0 \qquad para\ i \neq j \qquad (3)$$

Em que Ø representa os sinais que formam a base do diagrama. Para esta base ter alguma utilidade, todos os valores de amplitude e fase que a portadora pode assumir, devem ser representáveis nesta base, ou seja, consistem em combinações lineares dos eixos, respeitando assim a seguinte relação:

$$s_i(t) = \sum_{j=1}^{N} s_{ij}\emptyset_j(t) \quad (4)$$

Em que s_i corresponde aos sinais à serem representados em N eixos (RAPPAPORT, 2001, p. 236). Outra propriedade que os sinais candidatos à base devem satisfazer, está relacionada à sua energia, a qual deve possuir valor unitário. Traduzindo para a álgebra linear, esta propriedade garante que os eixos serão ortonormais:

$$E = \int_{-\infty}^{\infty} \emptyset_i{}^2(t)dt = 1 \quad (5)$$

Considerando agora os sinais da constelação mostrada anteriormente, pode-se definí-los matematicamente como:

$$S_1(t) = Acos(2\pi f_c t) \quad (6)$$

$$S_0(t) = Acos(2\pi f_c t + \pi) \quad (7)$$

Em que S_1 representa o sinal correspondente ao bit 1, e S_0 ao bit 0 (RAPPAPORT, 2001, p. 239). Como pode-se observar, ambos os sinais possuem a mesma amplitude, distantes 180° em fase, como sugere o diagrama. A amplitude do sinal pode ser definida em função da energia do bit, da seguinte forma. Primeiramente, em uma codificação digital unipolar, sabe-se que só se gasta energia nas transmissões de níveis lógicos altos, já que níveis lógicos baixos são essencialmente ausência de sinal. Portanto, supondo-se que em uma transmissão de dados considerável, o número de bits 1 e 0 são iguais, conclui-se que a energia média de um bit corresponde à metade da energia de um bit. A sequência abaixo resume o que foi dito.

$$E_b = Energia\ média\ de\ um\ bit = \frac{n*0+n*E}{2n} = \frac{E}{2} \quad (8)$$

Assim, tem-se a amplitude em função da energia média de um bit respeitando a seguinte relação:

$$P_r = A^2 \tag{9}$$

$$P_r = \frac{E}{T_b} = \frac{2E_b}{T_b} \tag{10}$$

$$A = \sqrt{\frac{2E_b}{T_b}} \tag{11}$$

Portanto, pode-se reescrever os sinais (RAPPAPORT, 2001, p. 239):

$$S_1(t) = Acos(wt) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t) \tag{12}$$

$$S_0(t) = Acos(wt + \pi) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t + \pi) = -\sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t) \tag{13}$$

Há diversas formas de se encontrar uma base para representar um diagrama, e infinitas bases que possam representá-la satisfazendo as condições citadas acima. Como as técnicas para o cálculo das bases consiste em técnicas características de álgebra linear, não se entrará no mérito de explicá-las e deduzí-las. Portanto, assumindo um eixo horizontal com a seguinte características:

$$\emptyset_1 = \sqrt{\frac{2}{T_b}}\cos(2\pi f_c t) \tag{14}$$

Pode-se então representar os sinais do diagrama com a seguinte notação, a qual claramente tem como representação geométrica, o diagrama mostrado anteriormente (RAPPAPORT, 2001, p. 236):

$$S_{BPSK} = \{\sqrt{E_b}\emptyset_1(t), -\sqrt{E_b}\emptyset_1(t)\}$$

Para se entender como a modulação ocorre, deve-se estudar os processos contidos em um modulador BPSK:

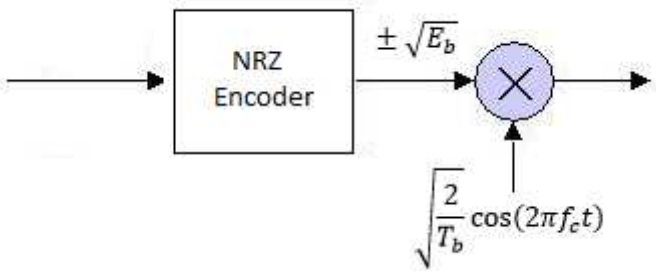


Figura 2 – Transmissão segundo esquema de modulação BPSK. Fonte: autor

A informação digital a ser transmitida passa por um *NRZ Encoder,* o qual transforma o sinal digital em bipolar, com amplitude igual à representada no diagrama, ou seja, igual a $\sqrt{E_b}$. Este *encoder* tem como característica exatamente o que seu nome sugere, ou seja, em sequências de níveis lógicos altos o sinal de saída mantêm o valor positivo de $\sqrt{E_b}$, sem retornar à zero (RAPPAPORT, 2001, p. 246). Após realizado a modulação, a onda é transmitida por uma antena.

Já um demodulador BPSK possui o seguinte formato:

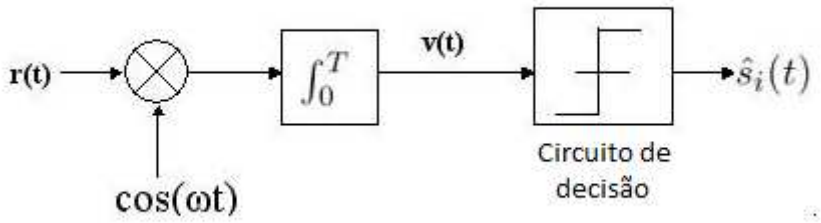


Figura 3 – Demodulador BPSK. Fonte: autor.

O sinal recebido pela antena é multiplicado por uma onda com a mesma frequência da portadora, recuperando o sinal transmitido. Este processo fica claro quando se analisa a seguinte propriedade trigonométrica (RAPPAPORT, 2001, p. 207):

$$\cos(2\pi f_c)\cos(2\pi f_c) = \frac{1}{2}(\cos(0) + \cos(4\pi f_c)) \quad (15)$$

Assim, ao se filtrar a componente de maior frequência gerada pelo produto, se recupera o sinal transmitido. Após recuperado, o circuito identificado como *Threshold Detector* compara o sinal recebido segundo o diagrama de constelação, para identificar qual bit foi originalmente transmitido. Como os sinais estão defasados em 180°, torna-se intuitivo delimitar uma região limite na origem do diagrama. Isto significa que caso o sinal que chegue no circuito de decisão seja maior que zero, o receptor identificará o bit transmitido como nível lógico alto, e o contrário caso o sinal seja inferior a zero (RAPPAPORT, 2001, p. 246).

A partir da caracterização do sistema de decisão, pode-se finalmente entender o cálculo do parâmetro BER de um método de modulação. Neste caso, o sistema terá erro caso se tenha transmitido um bit de nível lógico alto, e o sinal correspondente no circuito de decisão seja inferior a zero, ou ao transmitir um bit de nível lógico baixo, o sinal correspondente no circuito de decisão seja superior a zero. Obviamente, como já antecipado, isto é possível devido a interação do sinal com o ruído, o qual tem sua probabilidade aproximada para um comportamento gaussiano. A Fig. 4 auxilia na visualização do cálculo do BER (RAPPAPORT, 2001, p. 246):

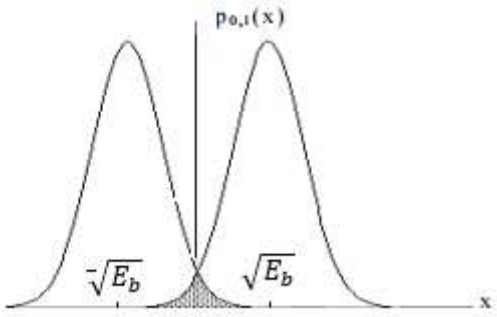


Figura 4 - Visualização do cálculo do BER para o método de modulação BPSK. Fonte (Moher e Haykin, 2011).

Ao olhar a figura pode-se observar que as distribuições de ruído foram somadas aos possíveis valores que a onda portadora pode assumir. A área hachurada mostra exatamente a situação em que o ruído age tão drasticamente no sinal, que o seu valor resultante ultrapassa a região limite, fazendo com que o circuito de decisão interprete erroneamente o bit transmitido. Para representar ambas as situações, faz-se necessário utilizar as seguintes probabilidades condicionadas (Moher e Haykin, 2011):

$$P\ (erro) = P(decisão\ bit\ 1 \mid 0\ transmitido) + P(decisão\ bit\ 0 \mid 1\ transmitido) \quad (16)$$

$$P\ (erro)\ \ = P\ (transm\ 1)P(errar \mid transm\ 1) + (transm\ 0)P(errar \mid transm\ 0) \quad (17)$$

Supondo que em uma transmissão significativa, os números de bits 1 e 0 são iguais (Moher e Haykin, 2011):

$$P\ (transmitido\ 1) = \ P\ (transmitido\ 0) = \frac{1}{2} \quad (18)$$

Calcula-se a probabilidade de errar tendo transmitido o bit 1:

$$P(errar \mid bit\ 1\ transmitido) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{0} e^{\frac{-\left(x-\sqrt{E_b}\right)^2}{2\sigma^2}}\,dx \quad (19)$$

Usando-se da função de erro complementar (Moher e Haykin, 2011):

$$P\ (errar\ |bit\ 1\ transmitido) = \frac{1}{2}erfc\left(\sqrt{\frac{E_b}{N_0}}\right) \quad (20)$$

Segundo a simetria do problema, segue-se os mesmos passos para o caso em que o bit 0 é transmitido. A equação seguinte mostra a equação final para o cálculo do BER (Moher e Haykin, 2011).

$$BER = P(errar) = \frac{1}{2} * \frac{1}{2}erfc\left(\sqrt{\frac{E_b}{N_0}}\right) + \frac{1}{2} * \frac{1}{2}erfc\left(\sqrt{\frac{E_b}{N_0}}\right) = \frac{1}{2}erfc\left(\sqrt{\frac{E_b}{N_0}}\right) \quad (21)$$

Esta fórmula indica o que já se havia concluído em relação à performance do sistema. Quanto maior for a relação sinal ruído, menor será a probabilidade de erro no circuito de decisão.

Elucidado sobre esta técnica mais simples de modulação, pode-se finalmente explicar as demais técnicas implementadas no programa.

1.2 QPSK

Este método de modulação, o qual leva o nome de *Quadrature Phase-Shift Keying,* consiste na modulação digital em fase da onda portadora, para a transmissão de dois bits por símbolo (RAPPAPORT, 2001, p. 246). Esta notação de bits por símbolo, é muito comum, e representa o grupo de bits que serão modulados na portadora. Normalmente, se usa a letra M para representar este parâmetro, a qual representa o número de sinais necessários para a representação do grupo de bits. Portanto, como exemplo, para esta modulação, se faz necessário quatro formas diferentes de deslocar em fase a portadora, para representar as quatro possíveis combinações, que a associação de dois bits pode assumir (RAPPAPORT, 2001, p. 221):

$$M = 2^{número\ de\ bits/símbolo} \tag{22}$$

O Diagrama de Constelação de um QPSK (RAPPAPORT, 2001, p. 249) pode ser visto na Fig.5.

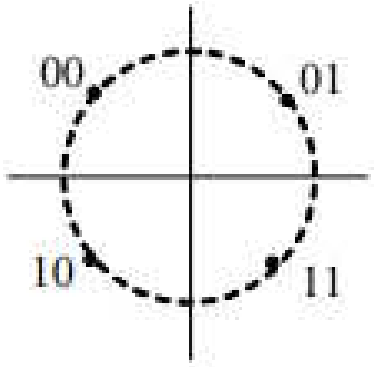


Figura 5 –Exemplo de um Diagrama de Constelação de um QPSK. Fonte: (Moher e Haykin, 2011).

Ao se observar o diagrama, nota-se que os possíveis valores que a portadora pode assumir, para representar o grupo de bits, possuem mesma amplitude, com fases distintas. Em contrapartida à técnica BPSK, neste caso, faz-se necessário duas dimensões para a representação dos valores que a portadora pode assumir. A partir do que já foi explanado sobre BPSK, torna-se mais fácil visualizar o processo de modulação com a Fig. 6.

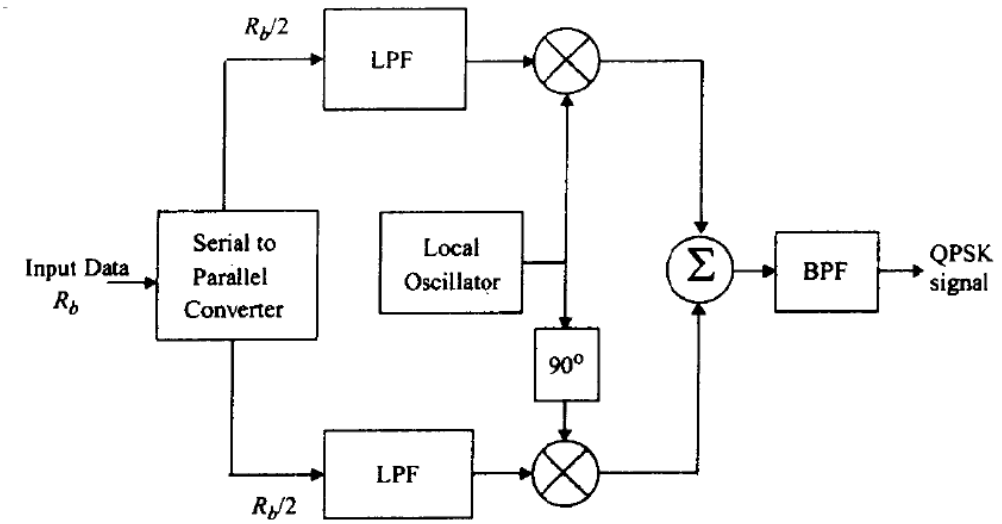


Figura 6 – Exemplo de um transmissor QPSK. Fonte: (RAPPAPORT, 2001, p. 247).

Primeiramente, o sinal discretizado a ser transmitido passa por um demultiplexador, no qual se divide metade dos bits para cada ramo. Os processos seguintes caracterizam os mesmos de uma modulação BPSK. Pode-se observar pela figura, que cada ramo recebe uma portadora, as quais são ortogonais entre si. Faz-se necessário este princípio da ortogonalidade, pois assim, ao somá-las, não há interferências entre as portadoras, e podem ser demoduladas separadamente no receptor (RAPPAPORT, 2001, p. 247).

O processo de demodulação possui o seguinte esquema (Fig.7):

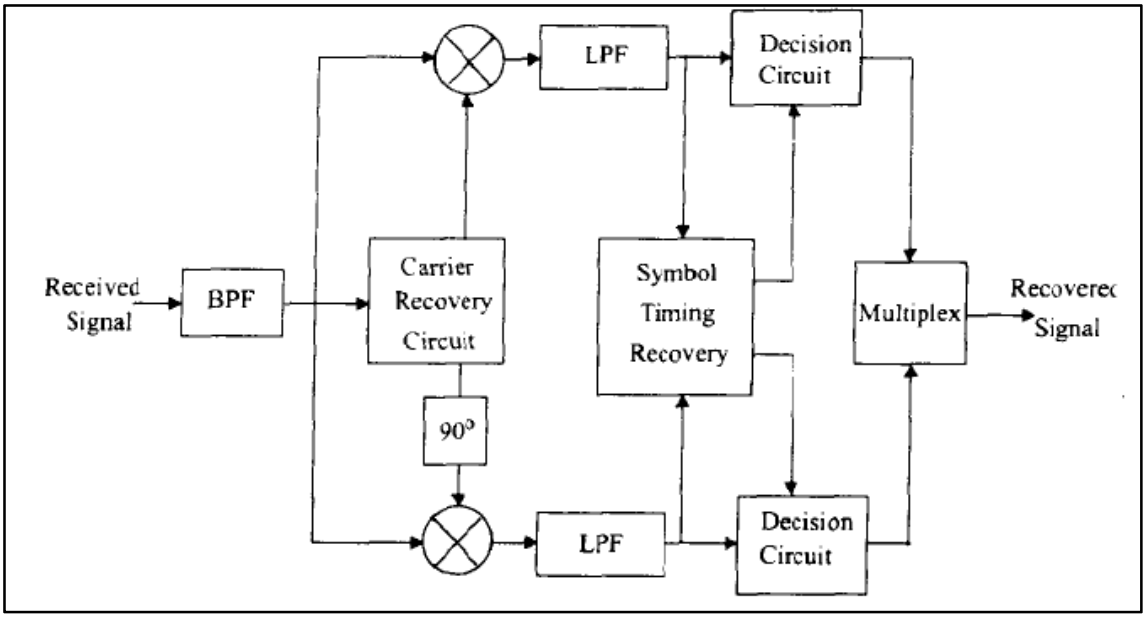


Figura 7 – Demodulador QPSK. Fonte (RAPPAPORT, 2001, p. 247).

Note-se que a demodulação para cada ramo usa-se de ondas defasadas 90°, como no processo de modulação. Pela ortogonalidade das ondas portadoras de cada ramo, só se faz possível demodular a sua respectiva base, já que o produto entre sinais ortogonais é sempre zero. Após demodulado cada ramo, o sinal passa pelo circuito de decisão, e em seguida usa-se de um multiplexador para reagrupar os bits. É importante ressaltar que neste caso a região limite é ligeiramente mais complexa, e se encontra exatamente sobre os eixos (RAPPAPORT, 2001, p. 246). Portanto, para exemplificar, o circuito de decisão interpretará o símbolo transmitido como "01" caso a fase se encontre entre 0° e 90°.

Como este método de modulação consiste de fato na soma de dois métodos BPSK ortogonais entre si, conclui-se que a probabilidade de erro do sistema é idêntica à já calculada (RAPPAPORT, 2001, p. 244):

$$P\ (erro) = \frac{1}{2} erfc\left(\sqrt{\frac{E_s}{N_0}}\right) \qquad (23)$$

Como neste caso, a transmissão se trata de dois bits por símbolo, tem-se que:

$$E_s = 2E_b \tag{24}$$

E portanto:

$$BER = P\,(erro) = \frac{1}{2} erfc\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{25}$$

A partir desta equação, em comparação com a fórmula do BER para o método BPSK, torna-se interessante introduzir uma discussão sobre o *trade-off* entre eficiência espectral e eficiência energética (RAPPAPORT, 2001, p. 222), o qual é de suma importância em projeto de sistemas de comunicação em geral. Com uma simples observação de ambas as fórmulas, é possível identificar que para um mesmo valor de BER, a modulação QPSK exige duas vezes mais potência de transmissão que o método BPSK. Em compensação, a modulação QPSK transmite duas vezes mais informação que a modulação BPSK, por apresentar dois bits por símbolo. Neste contexto, demonstra-se mais duas figuras de mérito importantes na escolha de um método de modulação, os quais são formalmente definidos a seguir (RAPPAPORT, 2001, p. 221):

$$eficiência\ espectral = \frac{bits/s}{Hz} \tag{26}$$

Este parâmetro consiste em uma razão entre o número de bits transmitidos por segundo e largura de banda. Já a eficiência energética consiste na própria razão sinal-ruído já discutida neste trabalho, sendo usada comparativamente considerando-se um valor de BER fixo.

No programa em questão implementou-se uma forma alternativa de QPSK que leva o nome de OQPSK (*Offset Quadrature Phase-Shift Keying*), o qual consiste basicamente no QPSK já citado, porém, o seu diferencial reside em um atraso intencional (um *offset*) em um dos ramos do modulador (um dos eixos da base), fazendo com a mudança de fase da portadora nunca passe pela origem do diagrama. Inversões bruscas de fase (180°) causam problemas no *hardware* que não serão abordados aqui.

A figura 8 mostra o diagrama de um OQPSK:

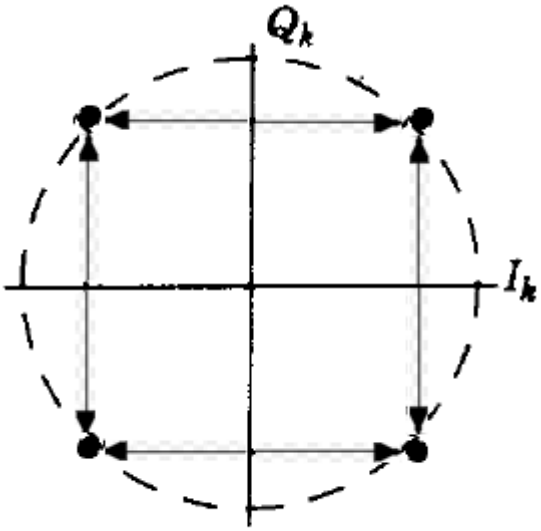


Figura 8 – Diagrama de constelação de um OQPSK. Fonte: adaptado de (RAPPAPORT, 2001, p. 244).

Como as setas indicam, se apenas um bit mudar, a portadora jamais sofrerá uma mudança brusca de 180°, mudando apenas 90°. Isto é possível atrasando intencionalmente um dos ramos da figura X, como já dito. Obviamente, o receptor deve levar em consideração este atraso para reconstruir corretamente a mensagem digital transmitida (RAPPAPORT, 2001, p. 247).

1.3 8PSK

Este método de modulação ainda se encontra no mesmo grupo dos explicados anteriormente, ou seja, consiste um uma prática de se modular a fase de uma onda portadora. Porém, neste caso, a transmissão possui três bits por símbolo.

O diagrama de constelação deste método se encontra na Fig.9.

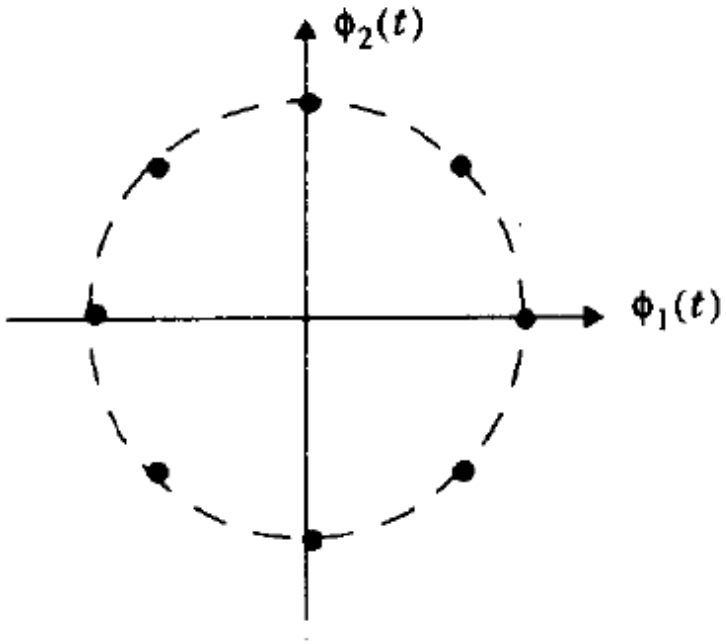


Figura 9 – Diagrama de constelação do método 8PSK. Fonte (RAPPAPORT, 2001, p. 268).

Os sinais a serem transmitidos podem ser resumidos em uma fórmula (RAPPAPORT, 2001, p. 267):

$$S_i(t) = \sqrt{\frac{2E_s}{T_s}}\cos\left[(i-1)\frac{\pi}{4}\right]\cos(2\pi f_c t) \; - \; \sqrt{\frac{2E_s}{T_s}}\sin\left[(i-1)\frac{\pi}{4}\right] sen(2\pi f_c t) \;\; i = 1,2,3 \ldots 8 \quad (27)$$

A região limite neste caso, pode ser visualizada imaginando-se raios que passam exatamente no ponto médio entre as distâncias dos sinais. Aliás, a partir das distâncias pode-se calcular a probabilidade de erro, ou em outras palavras, o parâmetro BER deste modo de modulação. Sabendo-se que as distâncias entre os sinais possuem o seguinte valor (RAPPAPORT, 2001, p. 268):

$$d = 2\sqrt{E_s} sen\left(\frac{\pi}{8}\right) \quad (28)$$

Calcula-se o BER (RAPPAPORT, 2001, p. 268):

$$BER = Probabiliade\ de\ erro = 2Q\left(\sqrt{\frac{6E_b}{N_0}}sen\left(\frac{\pi}{8}\right)\right) \quad (29)$$

Neste caso a dedução não foi feita já que para muitos bits por símbolo, a mesma se torna longa e complicada, porém, segue-se a mesma metodologia da usada para o método BPSK.

A tabela a seguir mostra a mudança entre eficiência espectral e energética com o aumento de bits por símbolos em modulações de fase:

Tabela 1 – Valores de eficiência espectral e energética com o aumento de bits por símbolo em modulação de fase. Fonte (RAPPAPORT, 2001, p. 269)

M	**2**	**4**	**8**	**16**	**32**	**64**
Eficiência Espectral	0,5	1	1,5	2	2,5	3
E_b/N_0 **para BER =** 10^{-6}	10,5	10,5	14	18,5	23,4	28,5

1.4 MQAM

Nas formas anteriores de modulação discutidas, modulava-se apenas a fase da portadora, resultando em diagramas de constelação circulares, já que a amplitude se mantinha constante. Neste método, modula-se a amplitude da portadora para a representação de grupos de bits.

A figura a seguir mostra o diagrama de constelação para diferentes bits por símbolo para uma modulação de amplitude em quadratura (RAPPAPORT, 2001, p. 270).

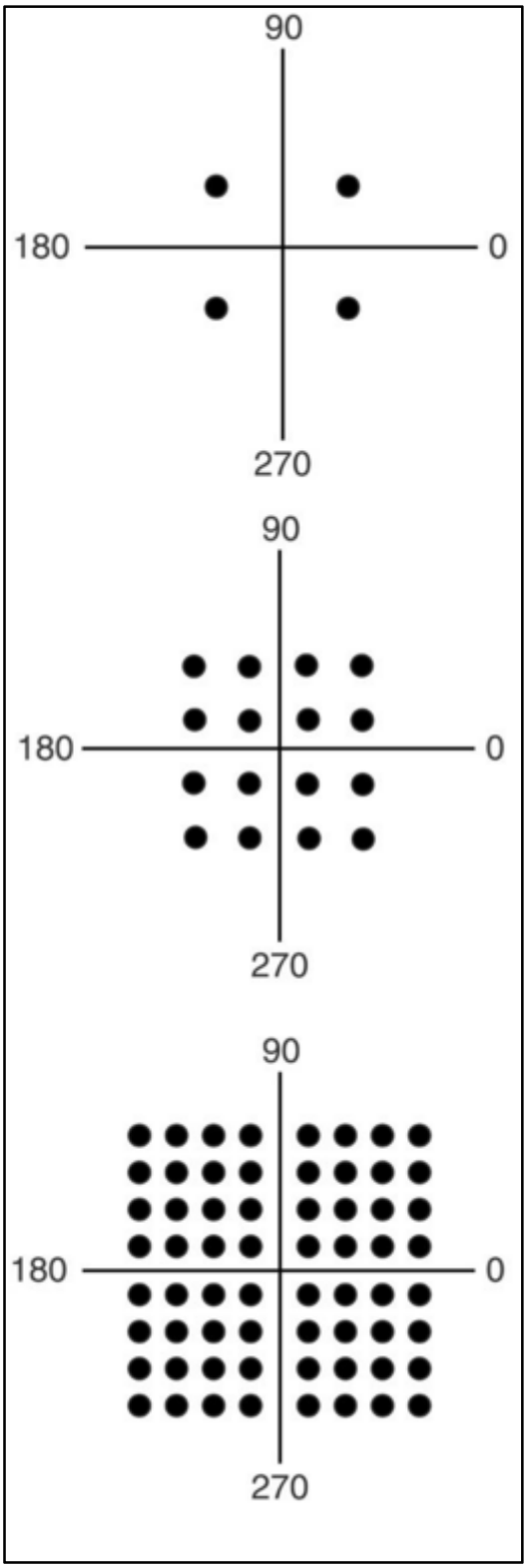


Figura 10 – Diagramas de constelação para 4QAM, 16QAM e 64QAM. Fonte: adaptado de (TILAL, 2014, p. 3).

A Fig. 10 mostra os diagramas de constelação para os métodos 4QAM, 16QAM E 64QAM. A fórmula geral para a composição dos sinais em função do número de bits por símbolo apresenta a seguinte forma (RAPPAPORT, 2001, p. 270):

$$S_i(t) = \sqrt{\frac{2E_{min}}{T_s}}\, a_i \cos(2\pi f_c t) + \sqrt{\frac{2E_{min}}{T_s}}\, b_i \operatorname{sen}(2\pi f_c t) \qquad i = 1,2,3,\dots,M \qquad (30)$$

Em que E_{min} corresponde à energia do sinal com menor amplitude, ou seja, mais próximo do centro, e os coeficientes a_i e b_i estão relacionados com a posição do sinal no diagrama. Pode-se encontrá-los da seguinte forma (RAPPAPORT, 2001, p. 271):

$$\{a_i, b_i\} = \begin{bmatrix} (-L+1, L-1) & (-L+3, L-1) & \dots & (L-1, L-1) \\ (-L+1, L-3) & (-L+3, L-3) & \vdots & (L-1, L-3) \\ \vdots & \vdots & \vdots & \vdots \\ (-L+1, -L+1) & (-L+3, -L+1) & \dots & (L-1, -L+1) \end{bmatrix} \qquad (31)$$

Em que L corresponde à:

$$L = \sqrt{M} \qquad (32)$$

Para exemplificar, considerando uma modulação 16QAM, em que L=4:

$$\{a_i, b_i\} = \begin{bmatrix} (-3,3) & (-1,3) & (1,3) & (3,3) \\ (-3,1) & (-1,1) & (1,1) & (3,1) \\ (-3,-1) & (-1,-1) & (1,-1) & (3,-1) \\ (-3,-3) & (-1,-3) & (1,-3) & (3,-3) \end{bmatrix} \qquad (33)$$

Algumas considerações são pertinentes ao se comparar estes diagramas com os de modulação em fase já apresentados. Primeiramente, nota-se que as distâncias dos pontos ao centro já não se mantem constante como anteriormente, o que naturalmente caracteriza a modulação em amplitude. Outra observação, neste caso não tão óbvia, se relaciona ás distâncias

entre os sinais no diagrama. Nas modulações em fase, nas quais o diagrama apresenta um aspecto circular, os sinais se encontram posicionados equidistantes no plano. Já nas modulações MQAM, a distância entre os pontos muda, o que impacta no cálculo do BER, já que a probabilidade de um sinal cruzar a região limite devido ao ruído, levando o sistema a errar, depende de sua posição no diagrama. O cálculo do erro em função do número bits por símbolo pode ser aproximado por (RAPPAPORT, 2001, p. 272):

$$BER = P\,(erro) = 4\left(1 - \frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3E_m}{(M-1)N_0}}\right) \qquad (34)$$

Em que E_m corresponde à energia média entre os sinais.

A tabela a seguir relaciona a eficiência espectral com a eficiência energética das modulações MQAM à medida que se aumenta o número de bits por símbolo.

Tabela 2 – Eficiência espectral e energética das modulações MQAM em função do número de bits por símbolo. Fonte (RAPPAPORT, 2001, p. 272)

M	**4**	**16**	**64**	**256**	**1024**	**4096**
Eficiência Espectral	1	2	3	4	5	6
E_b/N_0 **para BER =** 10^{-6}	10,5	15	18,5	24	28	33,5

Ao comparar esta tabela com a já apresentada referente à modulação em fase, pode-se observar que as modulações em amplitude apresentam em geral eficiência energética superior (RAPPAPORT, 2001, p. 272). Isto não significa que modulações em amplitude são melhores, já que também apresentam suas desvantagens, porém, se fez esta comparação para demonstrar ao leitor como essas figuras de mérito já citadas podem auxiliar na escolha de um método de modulação para sistemas de comunicação.

ANEXO 1 – Biblioteca de funções de interação com o usuário

1.1 Funções da figura de interface

```
function varargout = layout_tcc(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @layout_tcc_OpeningFcn, ...
                   'gui_OutputFcn',  @layout_tcc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before layout_tcc is made visible.
function layout_tcc_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to layout_tcc (see VARARGIN)
movegui(gcf,'center');
% Table for rain attenuation calculation
table = [{1;2;4;6;7;8;10;12;15;20;25;30;35;40} {0.0000387;0.000154;...
0.00065;0.00175;0.00301;0.00454;0.0101;0.0188;0.0367;0.0751;0.124;...
0.187;0.263;0.35} {0.912;0.963;1.121;1.308;1.332;1.327;1.276;1.217;...
1.154;1.099;1.061;1.021;0.979;0.939} {0.0000352;0.000138;0.000591;...
0.00155;0.00265;0.00395;0.00887;0.0168;0.0335;0.0601;0.113;0.167;...
0.233;0.310} {0.88;0.923;1.075;1.265;1.312;1.31;1.264;1.2;1.128;1.065...
;1.03;1;0.963;0.929}];
table = cell2table(table);
table.Properties.VariableNames = {'Freq_Carrier' 'K_Horizontal'...
    'Alfa_Horizontal' 'K_Vertical' 'Alfa_Vertical'};
handles.table = table;
% Choose default command line output for layout_tcc
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes layout_tcc wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = layout_tcc_OutputFcn(hObject, eventdata, handles)
```

```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

1.2 Funções relacionadas às coordenadas geográficas

```
function entrada_latitudeA_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_latitudeA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function entrada_latitudeA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_latitudeA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_longitudeA_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_longitudeA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function entrada_longitudeA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_longitudeA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_latitudeB_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_latitudeB (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




function entrada_latitudeB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_latitudeB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_longitudeB_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_longitudeB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




function entrada_longitudeB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_longitudeB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

1.3 Funções relacionadas ao botão Update Data

```
function update_data_Callback(hObject, eventdata, handles)
% hObject    handle to update_data (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.export_data,'Visible','On');
latA=str2num(get(handles.entrada_latitudeA,'String'));
lonA=str2num(get(handles.entrada_longitudeA,'String'));
latB=str2num(get(handles.entrada_latitudeB,'String'));
lonB=str2num(get(handles.entrada_longitudeB,'String'));
amostras=str2num(get(handles.entrada_amostras,'String'));
freq=str2num(get(handles.entrada_freq,'String'));
height_tx=str2num(get(handles.antenna_height_tx,'String'));
height_rx=str2num(get(handles.antenna_height_rx,'String'));
```

```
% Checking if input given are numbers
if (isempty(latA)||isempty(lonA)||isempty(latB)||isempty(lonB))
    errordlg('Error in notation, reenter the coordinates');
end
% Plotting the data
legend(handles.plano_planalt,'hide'); % hide legend
set(handles.plano_planalt,'NextPlot','replacechildren'); % replace plot
plot_satellite(latA,lonA,latB,lonB,handles.mapa_google);
[dist vec_altitude]=plot_planalt(latA,lonA,latB,lonB,amostras,freq,...
    height_tx,height_rx,handles.plano_planalt);
handles.dist = dist;
handles.vec_altitude = vec_altitude;
set(handles.mapa_google,'Tag','mapa_google');
set(handles.plano_planalt,'Tag','plano_planalt');
% Making planalt plot invisible
set(handles.plano_planalt,'Visible','off')
a=get(handles.plano_planalt,'children');
set(a,'Visible','off');
guidata(hObject,handles);
% Calculating technical parameters
run('tech_parameters.m')
% Generating text
run('Link_considerations.m')
```

1.4 Funções dos elementos referentes aos parâmetros do enlace

```
function entrada_freq_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_freq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function entrada_freq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_freq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_amostras_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_amostras (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
guidata(hObject,handles);



% --- Executes during object creation, after setting all properties.
```

```
function entrada_amostras_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_amostras (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_EIRP_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_EIRP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function entrada_EIRP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_EIRP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_Grecep_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_Grecep (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function entrada_Grecep_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_Grecep (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function entrada_sens_recep_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to entrada_sens_recep (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function entrada_sens_recep_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_sens_recep (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function output_rain_Callback(hObject, eventdata, handles)
% hObject    handle to output_rain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function output_rain_CreateFcn(hObject, eventdata, handles)
% hObject    handle to output_rain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
```

```
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in mudar_plot.
```

1.5 Funções relacionadas ao botão Plot Planalt/Google Maps

```
function mudar_plot_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to mudar_plot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(handles.plano_planalt.Visible,'on');
    set(handles.plano_planalt,'Visible','off');
    a=get(handles.plano_planalt,'children');
    set(a,'Visible','off');
    legend(handles.plano_planalt,'hide');
    set(handles.mapa_google,'Visible','on');
    guidata(hObject,handles);
else
    set(handles.plano_planalt,'Visible','on');
    legend(handles.plano_planalt,'Terrain','Terrain considering curvature');
    a=get(handles.plano_planalt,'children');
    set(a,'Visible','on');
    set(handles.mapa_google,'Visible','off');
    guidata(hObject,handles);
end
```

1.6 Funções relacionadas aos outros elementos gráficos

```
function output_freespace_Callback(hObject, eventdata, handles)
% hObject    handle to output_freespace (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function output_freespace_CreateFcn(hObject, eventdata, handles)
% hObject    handle to output_freespace (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function sens_rx_Callback(hObject, eventdata, handles)
% hObject    handle to sens_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function sens_rx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sens_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function antenna_gain_rx_Callback(hObject, eventdata, handles)
% hObject    handle to antenna_gain_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function antenna_gain_rx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenna_gain_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
```

```
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function antenna_height_rx_Callback(hObject, eventdata, handles)
% hObject    handle to antenna_height_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




% --- Executes during object creation, after setting all properties.
function antenna_height_rx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenna_height_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function antenna_gain_tx_Callback(hObject, eventdata, handles)
% hObject    handle to antenna_gain_tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function antenna_gain_tx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenna_gain_tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function antenna_height_tx_Callback(hObject, eventdata, handles)
% hObject    handle to antenna_height_tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function antenna_height_tx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenna_height_tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function power_tx_Callback(hObject, eventdata, handles)
% hObject    handle to power_tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function power_tx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to power_tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit25_Callback(hObject, eventdata, handles)
% hObject    handle to edit25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit25_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

1.7 Funções relacionadas aos botões de modulação

```
function ber_calc_Callback(hObject, eventdata, handles)
% hObject    handle to ber_calc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function ber_calc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ber_calc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
```

```
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in qpsk.
function qpsk_Callback(hObject, eventdata, handles)
% hObject    handle to qpsk (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tent = get(hObject,'Tag');

% Hint: get(hObject,'Value') returns toggle state of qpsk


% --- Executes on button press in eight_psk.
function eight_psk_Callback(hObject, eventdata, handles)
% hObject    handle to eight_psk (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 if isfield(handles,'SNR_dB')
    ber = berawgn(SNR_dB,'psk',8,'nondiff');
    set(handles.ber_calc,'String',num2str(ber));
 end
% Hint: get(hObject,'Value') returns toggle state of eight_psk


% --- Executes on button press in sixteen_qam.
function sixteen_qam_Callback(hObject, eventdata, handles)
% hObject    handle to sixteen_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of sixteen_qam


% --- Executes on button press in thirtytwo_qam.
function thirtytwo_qam_Callback(hObject, eventdata, handles)
% hObject    handle to thirtytwo_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of thirtytwo_qam


% --- Executes on button press in sixtyfour_qam.
function sixtyfour_qam_Callback(hObject, eventdata, handles)
% hObject    handle to sixtyfour_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of sixtyfour_qam


% --- Executes on button press in onetwentyeight_qam.
function onetwentyeight_qam_Callback(hObject, eventdata, handles)
% hObject    handle to onetwentyeight_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of onetwentyeight_qam


% --- Executes on button press in twofiftysix_qam.
function twofiftysix_qam_Callback(hObject, eventdata, handles)
% hObject    handle to twofiftysix_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of twofiftysix_qam


% --- Executes on button press in fivetwelve_qam.
function fivetwelve_qam_Callback(hObject, eventdata, handles)
% hObject    handle to fivetwelve_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of fivetwelve_qam


% --- Executes on button press in tentwentyfour_qam.
function tentwentyfour_qam_Callback(hObject, eventdata, handles)
% hObject    handle to tentwentyfour_qam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of tentwentyfour_qam
```

1.8 Funções de outros elementos

```
function bandwidth_Callback(hObject, eventdata, handles)
% hObject    handle to bandwidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function bandwidth_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bandwidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function elev_angle_Callback(hObject, eventdata, handles)
% hObject    handle to elev_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function elev_angle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to elev_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function noise_figure_Callback(hObject, eventdata, handles)
% hObject    handle to noise_figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function noise_figure_CreateFcn(hObject, eventdata, handles)
% hObject    handle to noise_figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function antenna_temp_rx_Callback(hObject, eventdata, handles)
% hObject    handle to antenna_temp_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function antenna_temp_rx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to antenna_temp_rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on selection change in listbox.
function listbox_Callback(hObject, eventdata, handles)
% hObject    handle to listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

1.9 Funções relacionadas ao botão Export Data

```
function export_data_Callback(hObject, eventdata, handles)
% hObject    handle to export_data (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(handles.plano_planalt.Visible,'off');
    mudar_plot_Callback(hObject,eventdata,handles);
end
run('export_data.m');



function precipitation_Callback(hObject, eventdata, handles)
% hObject    handle to precipitation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function precipitation_CreateFcn(hObject, eventdata, handles)
% hObject    handle to precipitation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function link_margin_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to link_margin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function link_margin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to link_margin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Published with MATLAB® R2015a

ANEXO 2 – Função para realizar o plano planaltimétrico

2.1 Comunicação com o API google

```
function [dist vec_altitude] = plot_planalt(lat1,lon1,lat2,lon2,...
        amostras,f,height_tx,height_rx,handles)
```

```
    lat(1)=lat1;
    lat(2)=lat2;
    lon(1)=lon1;
    lon(2)=lon2;
    url_p1='http://maps.googleapis.com/maps/api/elevation/xml?path=';
    url_p2='&sensor=true_or_false';
    amostras_str = num2str(amostras);
    lat_str=arrayfun(@(x) num2str(x),lat,'Un',0);
    lon_str=arrayfun(@(x) num2str(x),lon,'Un',0);
    url=[url_p1 lat_str{1} ',' lon_str{1} '|' lat_str{2} ',' lon_str{2}...
        '&samples=' amostras_str url_p2];
     texto=urlread(url);
```

2.2 Adequando os dados

```
     texto=char2cell(texto,'>');
     texto=Retorna_Linhas(texto,'</elevation');
     texto=cellfun(@(x) strsplit(x,'<'),texto,'Un',0);
     for i=1:amostras
         texto_aux{i}=texto{i}{1};
         vec_altitude(i)=str2num(texto_aux{i});
     end
        dist = varargout(lat(1),lon(1),lat(2),lon(2));
        dist=linspace(0,dist,amostras);
        dist=dist/1000;
```

2.3 Considerando a curvatura da terra

```
    curvature;
    vec_altitude_curv = 1000.*y + vec_altitude;
```

2.4 Realizando os gráficos

```
    plot(handles,dist,vec_altitude);
    set(handles,'NextPlot','add'); %set plot to add children
    plot(handles,dist,vec_altitude_curv);
    set(handles,'YLim',[0 1.5*max(vec_altitude_curv)]); %limits
    set(handles,'XLim',[0 dist(end)]);
    dist=dist'; % Preparing data for fresnel plot
    vec_altitude = vec_altitude';
```

2.5 Gráfico da primeira Zona de Fresnel

```
    [xis,altura1,altura2]=fresnel(height_tx,height_rx,f,dist,...
        vec_altitude);
    plot(handles,dist,altura1);
    plot(handles,dist,altura2);
```

```
end
```

Published with MATLAB® R2015a

ANEXO 3 – Cálculo da curvatura da terra usando um raio médio

```
x1 = 0;
x2 = dist(end);
y1 = 0; % Setting the limits of the arc
y2 = 0;
r = earthRadius/1000;
d = sqrt((x2-x1)^2+(y2-y1)^2); % Distance between points
a = atan2(x2-x1,-(y2-y1));
b = asin(d/2/r); % Half arc angle
c = linspace(a-b,a+b,amostras); % Arc angle range
e = sqrt(r^2-d^2/4); % Distance, center to midpoint
x = (x1+x2)/2-e*cos(a)+r*cos(c); % Cartesian coords. of arc
y = (y1+y2)/2-e*sin(a)+r*sin(c);

% Roger Stafford
```

Published with MATLAB® R2015a

ANEXO 4 – Função para o cálculo da primeira Zona de Fresnel

```
function [xis,altura1, altura2] = fresnel(antena1, antena2,f,...
    distancias,elevacao)
% First Fresnel Zone Calculation

distancias = distancias*1000;
q=1;
lambda = 3e8/(f);
alfa= atan((elevacao(1)+antena1-elevacao(size(elevacao,1))-antena2)/...
    (distancias(size(distancias,1))-distancias(1)));
d1 = distancias(1) ;
e1 = elevacao(1);
distancias = distancias-distancias(1);
elevacao = elevacao-elevacao(1);
xis =  distancias*cos(alfa);
altura = (q*(lambda*(distancias).*(distancias(size(distancias,1))-...
    distancias))./(distancias(size(distancias,1)))).^0.5;
altura1 = -xis*sin(alfa) - altura*cos(alfa)+e1+antena1;
altura2 = -xis*sin(alfa) + altura*cos(alfa)+e1+antena1;



end
```

Published with MATLAB® R2015a

ANEXO 5 – Função para a visualização de satélite do enlace

```
function [ dist,vec_altitude ] = plot_satellite(lat1,lon1,lat2,lon2,...
    handles)
```

```
%   This function communicates to Googles API
   lat(1)=lat1;
   lat(2)=lat2;
   lon(1)=lon1;
   lon(2)=lon2;
   plot(handles,lon,lat,'.r','MarkerSize',20)
   hold on;
```

5.1 Chamando a função que realiza a comunicação com o API

```
  googlemap('axis',handles);
```

5.2 Desenhenado uma linha entre os pontos

```
   y_coef=polyfit(lat,lon,1);
   x=linspace(lat(1),lat(2),100);
   y=y_coef(1)*x+y_coef(2);
   plot(y,x,'-r');
   hold off;
```

```
end
```

Published with MATLAB® R2015a

ANEXO 6 – Função para a comunicação com o API Google

```
function [varargout] = googlemap(varargin)
```

```
% function h = plot_google_map(varargin)
% Plots a google map on the current axes using the Google Static Maps API
```

6.1 Preparando o API

```
persistent apiKey
if isnumeric(apiKey)
    % first run, check if API key file exists
    if exist('api_key.mat','file')
        load api_key
    else
        apiKey = '';
    end
end
hold on
```

6.2 Parâmetros gráficos

```
axHandle = gca;
height = 640;
width = 640;
scale = 2;
maptype='hybrid';
alphaData = 1;
autoRefresh = 1;
figureResizeUpdate = 1;
autoAxis = 1;
showLabels = 1;
language = '';
markeridx = 1;
markerlist = {};

% Handle input arguments
if nargin >= 2
    for idx = 1:2:length(varargin)
        switch lower(varargin{idx})
            case 'axis'
                axHandle = varargin{idx+1};
            case 'height'
                height = varargin{idx+1};
            case 'width'
                width = varargin{idx+1};
            case 'maptype'
                maptype = varargin{idx+1};
            case 'alpha'
                alphaData = varargin{idx+1};
            case 'refresh'
                autoRefresh = varargin{idx+1};
            case 'showlabels'
                showLabels = varargin{idx+1};
            case 'figureresizeupdate'
```

```
                figureResizeUpdate = varargin{idx+1};
            case 'language'
                language = varargin{idx+1};
            case 'marker'
                markerlist{markeridx} = varargin{idx+1};
                markeridx = markeridx + 1;
            case 'autoaxis'
                autoAxis = varargin{idx+1};
            case 'apikey'
                apiKey = varargin{idx+1}; % set new key
                % save key to file
                funcFile = which('plot_google_map.m');
                pth = fileparts(funcFile);
                keyFile = fullfile(pth,'api_key.mat');
                save(keyFile,'apiKey')
            otherwise
                error(['Unrecognized variable: ' varargin{idx}])
        end
    end
end
if height > 640
    height = 640;
end
if width > 640
    width = 640;
end

% Store paramters in axis handle (for auto refresh callbacks)
ud = get(axHandle, 'UserData');
ud.gmap_params = varargin;
set(axHandle, 'UserData', ud);

curAxis = axis(axHandle);
% Enforce Latitude constraints of EPSG:900913
if curAxis(3) < -85
    curAxis(3) = -85;
end
if curAxis(4) > 85
    curAxis(4) = 85;
end
% Enforce longitude constrains
if curAxis(1) < -180
    curAxis(1) = -180;
end
if curAxis(1) > 180
    curAxis(1) = 0;
end
if curAxis(2) > 180
    curAxis(2) = 180;
end
if curAxis(2) < -180
    curAxis(2) = 0;
end

if isequal(curAxis,[0 1 0 1]) % probably an empty figure
    % display world map
    curAxis = [-200 200 -85 85];
    axis(curAxis)
```

```
end


if autoAxis
    % adjust current axis limit to avoid strectched maps
    [xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2) );
    xExtent = diff(xExtent); % just the size of the span
    yExtent = diff(yExtent);
    % get axes aspect ratio
    drawnow
    org_units = get(axHandle,'Units');
    set(axHandle,'Units','Pixels')
    ax_position = get(axHandle,'position');
    set(axHandle,'Units',org_units)
    aspect_ratio = ax_position(4) / ax_position(3);

    if xExtent*aspect_ratio > yExtent
        centerX = mean(curAxis(1:2));
        centerY = mean(curAxis(3:4));
        spanX = (curAxis(2)-curAxis(1))/2;
        spanY = (curAxis(4)-curAxis(3))/2;

        % enlarge the Y extent
        spanY = spanY*xExtent*aspect_ratio/yExtent; % new span
        if spanY > 85
            spanX = spanX * 85 / spanY;
            spanY = spanY * 85 / spanY;
        end
        curAxis(1) = centerX-spanX;
        curAxis(2) = centerX+spanX;
        curAxis(3) = centerY-spanY;
        curAxis(4) = centerY+spanY;
    elseif yExtent > xExtent*aspect_ratio

        centerX = mean(curAxis(1:2));
        centerY = mean(curAxis(3:4));
        spanX = (curAxis(2)-curAxis(1))/2;
        spanY = (curAxis(4)-curAxis(3))/2;
        % enlarge the X extent
        spanX = spanX*yExtent/(xExtent*aspect_ratio); % new span
        if spanX > 180
            spanY = spanY * 180 / spanX;
            spanX = spanX * 180 / spanX;
        end

        curAxis(1) = centerX-spanX;
        curAxis(2) = centerX+spanX;
        curAxis(3) = centerY-spanY;
        curAxis(4) = centerY+spanY;
    end
    % Enforce Latitude constraints of EPSG:900913
    if curAxis(3) < -85
        curAxis(3:4) = curAxis(3:4) + (-85 - curAxis(3));
    end
    if curAxis(4) > 85
        curAxis(3:4) = curAxis(3:4) + (85 - curAxis(4));
    end
    axis(axHandle, curAxis); % update axis as quickly as possible
```

```
    drawnow
end

% Delete previous map from plot (if exists)
if nargout <= 1 % only if in plotting mode
    curChildren = get(axHandle,'children');
    map_objs = findobj(curChildren,'tag','gmap');
    bd_callback = [];
    for idx = 1:length(map_objs)
        if ~isempty(get(map_objs(idx),'ButtonDownFcn'))
            % copy callback properties from current map
            bd_callback = get(map_objs(idx),'ButtonDownFcn');
        end
    end
    delete(map_objs)

end
```

6.3 Calculando zoom ótimo

```
[xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2) );
minResX = diff(xExtent) / width;
minResY = diff(yExtent) / height;
minRes = max([minResX minResY]);
tileSize = 256;
initialResolution = 2 * pi * 6378137 / tileSize;
zoomlevel = floor(log2(initialResolution/minRes));

% Enforce valid zoom levels
if zoomlevel < 0
    zoomlevel = 0;
end
if zoomlevel > 19
    zoomlevel = 19;
end

% Calculate center coordinate in WGS1984
lat = (curAxis(3)+curAxis(4))/2;
lon = (curAxis(1)+curAxis(2))/2;
```

6.4 Construindo o URL para acesso

```
preamble = 'http://maps.googleapis.com/maps/api/staticmap';
location = ['?center=' num2str(lat,10) ',' num2str(lon,10)];
zoomStr = ['&zoom=' num2str(zoomlevel)];
sizeStr = ['&scale=' num2str(scale) '&size=' num2str(width) 'x' num2str(height)];
maptypeStr = ['&maptype=' maptype ];
if ~isempty(apiKey)
    keyStr = ['&key=' apiKey];
else
    keyStr = '';
end
markers = '&markers=';
for idx = 1:length(markerlist)
    if idx < length(markerlist)
        markers = [markers markerlist{idx} '%7C'];
```

```
    else
        markers = [markers markerlist{idx}];
    end
end
if showLabels == 0
    labelsStr = '&style=feature:all|element:labels|visibility:off';
else
    labelsStr = '';
end
if ~isempty(language)
    languageStr = ['&language=' language];
else
    languageStr = '';
end

if ismember(maptype,{'satellite','hybrid'})
    filename = 'tmp.jpg';
    format = '&format=jpg';
    convertNeeded = 0;
else
    filename = 'tmp.png';
    format = '&format=png';
    convertNeeded = 1;
end
sensor = '&sensor=false';
url = [preamble location zoomStr sizeStr maptypeStr format markers ...
    labelsStr languageStr sensor keyStr];
```

6.5 Recebendo a imagem do API

```
try
    urlwrite(url,filename);
catch % error downloading map
    warning(sprintf(['Unable to download map form Google Servers.\n' ...
        'Possible reasons: no network connection, quota exceeded, or ...'
        'some other error.\n' ...
        'Consider using an API key if quota problems persist.\n\n' ...
        'To debug, try pasting the following URL in your browser, '...
        'which may result in a more informative error:\n%s'], url));
    varargout{1} = [];
    varargout{2} = [];
    varargout{3} = [];
    return
end
[M Mcolor] = imread(filename);
M = cast(M,'double');
delete(filename); % delete temp file
width = size(M,2);
height = size(M,1);

% Calculate a meshgrid of pixel coordinates in EPSG:900913
centerPixelY = round(height/2);
centerPixelX = round(width/2);
[centerX,centerY] = latLonToMeters(lat, lon );
curResolution = initialResolution / 2^zoomlevel/scale;
xVec = centerX + ((1:width)-centerPixelX) * curResolution; % x vector
yVec = centerY + ((height:-1:1)-centerPixelY) * curResolution; % y vector
```

```
[xMesh,yMesh] = meshgrid(xVec,yVec); % construct meshgrid

% convert meshgrid to WGS1984
[lonMesh,latMesh] = metersToLatLon(xMesh,yMesh);

% We now want to convert the image from a colormap image with an uneven
% mesh grid, into an RGB truecolor image with a uniform grid.
% This would enable displaying it with IMAGE, instead of PCOLOR.
% Advantages are:
% 1) faster rendering
% 2) makes it possible to display together with other colormap annotations

% Convert image from colormap type to RGB truecolor (if PNG is used)
if convertNeeded
    imag = zeros(height,width,3);
    for idx = 1:3
        imag(:,:,idx) = reshape(Mcolor(M(:)+1+(idx-1)*size(Mcolor,1)),...
            height,width);
    end
else
    imag = M/255;
end

% Next, project the data into a uniform WGS1984 grid
sizeFactor = 1; % factoring of new image
uniHeight = round(height*sizeFactor);
uniWidth = round(width*sizeFactor);
latVect = linspace(latMesh(1,1),latMesh(end,1),uniHeight);
lonVect = linspace(lonMesh(1,1),lonMesh(1,end),uniWidth);
[uniLonMesh,uniLatMesh] = meshgrid(lonVect,latVect);
uniImag = zeros(uniHeight,uniWidth,3);

uniImag =  myTurboInterp2(lonMesh,latMesh,imag,uniLonMesh,uniLatMesh);

if nargout <= 1 % plot map
    % display image
    hold(axHandle, 'on');
    h = image(lonVect,latVect,uniImag, 'Parent', axHandle);
    set(axHandle,'YDir','Normal')
    set(h,'tag','gmap')
    set(h,'AlphaData',alphaData)

    % add a dummy image to allow pan/zoom out to x2 of the image extent
    h_tmp = image(lonVect([1 end]),latVect([1 end]),zeros(2),...
        'Visible','off', 'Parent', axHandle);
    set(h_tmp,'tag','gmap')

    % older version (display without conversion to uniform grid)
    % h =pcolor(lonMesh,latMesh,(M));
    % colormap(Mcolor)
    % caxis([0 255])
    % warning off % to avoid strange rendering warnings
    % shading flat

    uistack(h,'bottom')
    axis(axHandle, curAxis) % restore original zoom
    if nargout == 1
        varargout{1} = h;
```

```
    end

    % if auto-refresh mode - override zoom callback to allow autumatic
    % refresh of map upon zoom actions.
    figHandle = axHandle;
    while ~strcmpi(get(figHandle, 'Type'), 'figure')
        % Recursively search for parent figure in case axes are in a panel
        figHandle = get(figHandle, 'Parent');
    end

    zoomHandle = zoom(axHandle);
    panHandle = pan(figHandle);
    if autoRefresh
        set(zoomHandle,'ActionPostCallback',@update_google_map);
        set(panHandle, 'ActionPostCallback', @update_google_map);
    else % disable zoom override
        set(zoomHandle,'ActionPostCallback',[]);
        set(panHandle, 'ActionPostCallback',[]);
    end

    % set callback for figure resize function, to update extents if figure
    % is streched.
    if figureResizeUpdate &&isempty(get(figHandle, 'ResizeFcn'))
        % set only if not already set by someone else
        set(figHandle, 'ResizeFcn', @update_google_map_fig);
    end

    % set callback properties
    set(h,'ButtonDownFcn',bd_callback);
else % don't plot, only return map
    varargout{1} = lonVect;
    varargout{2} = latVect;
    varargout{3} = uniImag;
end
```

6.6 Declaração das funções utilizadas

```
function [lon,lat] = metersToLatLon(x,y)
% Converts XY point from Spherical Mercator EPSG:900913 to lat/lon
originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
lon = (x ./ originShift) * 180;
lat = (y ./ originShift) * 180;
lat = 180 / pi * (2 * atan( exp( lat * pi / 180)) - pi / 2);

function [x,y] = latLonToMeters(lat, lon )
% Converts given lat/lon in WGS84 Datum to XY in Spherical Mercator
originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
x = lon * originShift / 180;
y = log(tan((90 + lat) * pi / 360 )) / (pi / 180);
y = y * originShift / 180;


function ZI = myTurboInterp2(X,Y,Z,XI,YI)
% An extremely fast nearest neighbour 2D interpolation, assuming both input
% and output grids consist only of squares, meaning:
% - uniform X for each column
% - uniform Y for each row
```

```
XI = XI(1,:);
X = X(1,:);
YI = YI(:,1);
Y = Y(:,1);

xiPos = nan*ones(size(XI));
xLen = length(X);
yiPos = nan*ones(size(YI));
yLen = length(Y);
% find x conversion
xPos = 1;
for idx = 1:length(xiPos)
    if XI(idx) >= X(1) && XI(idx) <= X(end)
        while xPos < xLen && X(xPos+1)<XI(idx)
            xPos = xPos + 1;
        end
        diffs = abs(X(xPos:xPos+1)-XI(idx));
        if diffs(1) < diffs(2)
            xiPos(idx) = xPos;
        else
            xiPos(idx) = xPos + 1;
        end
    end
end
% find y conversion
yPos = 1;
for idx = 1:length(yiPos)
    if YI(idx) <= Y(1) && YI(idx) >= Y(end)
        while yPos < yLen && Y(yPos+1)>YI(idx)
            yPos = yPos + 1;
        end
        diffs = abs(Y(yPos:yPos+1)-YI(idx));
        if diffs(1) < diffs(2)
            yiPos(idx) = yPos;
        else
            yiPos(idx) = yPos + 1;
        end
    end
end
ZI = Z(yiPos,xiPos,:);


function update_google_map(obj,evd)
% callback function for auto-refresh
drawnow;
try
    axHandle = evd.Axes;
catch ex
    % Event doesn't contain the correct axes. Panic!
    axHandle = gca;
end
ud = get(axHandle, 'UserData');
if isfield(ud, 'gmap_params')
    params = ud.gmap_params;
    plot_google_map(params{:});
end
```

```
function update_google_map_fig(obj,evd)
% callback function for auto-refresh
drawnow;
axes_objs = findobj(get(gcf,'children'),'type','axes');
for idx = 1:length(axes_objs)
    if ~isempty(findobj(get(axes_objs(idx),'children'),'tag','gmap'));
        ud = get(axes_objs(idx), 'UserData');
        if isfield(ud, 'gmap_params')
            params = ud.gmap_params;
        else
            params = {};
        end

        % Add axes to inputs if needed
        if ~sum(strcmpi(params, 'Axis'))
            params = [params, {'Axis', axes_objs(idx)}];
        end
        plot_google_map(params{:});
    end
end

%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
```

Published with MATLAB® R2015a

ANEXO 7 – Cálculo dos parâmetros do enlace

7.1 Atenuações

```
lambda = physconst('LightSpeed')/freq;
fspace_loss = fspl(1000*handles.dist(end),lambda); % Free Space Loss
aux = num2str(fspace_loss);
set(handles.output_freespace,'String',aux);
clear aux;
run('Rain.m'); % ITU Rain attenuation
```

7.2 Calculando o SNR

```
B=str2num(get(handles.bandwidth,'String')); % receiving the inputs
Pt=str2num(get(handles.power_tx,'String'));
sens_rx=str2num(get(handles.sens_rx,'String'));
L_freespace=str2num(get(handles.output_freespace,'String'));
NF=str2num(get(handles.noise_figure,'String'));
Ant_Temp=str2num(get(handles.antenna_temp_rx,'String'));
L_rain=handles.lrain;
Gt=str2num(get(handles.antenna_gain_tx,'String'));
Gr=str2num(get(handles.antenna_gain_rx,'String'));
link_margin=str2num(get(handles.link_margin,'String'));
Teq = Ant_Temp + (290*(power(10,NF/10) - 1) - 1)/power(10,Gr/10);
handles.Teq = Teq; % equivalent temp noise
Pr_dBm =  Pt + Gt + Gr - L_freespace - link_margin - L_rain; %power
handles.Pr_dBm = Pr_dBm;
Pr = power(10,Pr_dBm/10)*1e-3;
handles.Pr = Pr;
SNR = Pr/(physconst('Boltzmann')*Teq*B);
handles.SNR = SNR;
if ~isempty(get(handles.sens_rx,'String'))
    if sens_rx>Pr_dBm
        warndlg('Received Power is lower than receivers sensibility'...
            ,'Warning');
        movegui(gcf,'center');
    end
end
```

7.3 Calculando BER

```
aux = get(handles.modulation_panel.SelectedObject,'Tag')

switch aux
    case 'qpsk'
        %berqpsk
        EbNo = SNR/2;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'oqpsk','nondiff');
    case 'eight_psk'
        %ber8psk
        EbNo = SNR/3;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'psk',8,'nondiff');
    case 'sixteen_qam'
```

```
        %ber16qam
        EbNo = SNR/4;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',16)
    case 'thirtytwo_qam'
        %ber32qam
        EbNo = SNR/5;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',32)
    case 'sixtyfour_qam'
        %ber64qam
        EbNo = SNR/6;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',64)
    case 'onetwentyeight_qam'
        %ber128qam
        EbNo = SNR/7;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',128)
    case 'twofiftysix_qam'
        %ber256qam
        EbNo = SNR/8;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',256)
    case 'fivetwelve_qam'
        %ber512qam
        EbNo = SNR/9;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',512)
    case 'tentwentyfour_qam'
        %ber1024qam
        EbNo = SNR/10;
        EbNo_dB = 10*log10(EbNo);
        ber = berawgn(EbNo_dB,'qam',1024)
end
set(handles.ber_calc,'String',num2str(ber)); % Making BER visible
```

Published with MATLAB® R2015a

ANEXO 8 – Cálculo da atenuação de chuva

8.1 Testando se usuário forneceu precipitação

```
guidata(hObject,handles); %updating handles
if isempty(get(handles.precipitation,'String'))
    % Communication with Open Weather API
    flag_rain = 1;
    url_1 = 'http://api.openweathermap.org/data/2.5/forecast/daily?lat={';
    latitude = num2str(latA);
    longitude = num2str(lonA);
    url_2 = '}&lon={';
    url_3 = '}&cnt=16&mode=json';
    url_final = [url_1 latitude url_2 longitude url_3];
    S = webread(url_final)
    cels = S.list;
    clear S;
    for i=1:16
        if(isfield(cels{i,1},'rain'))
            rain(i) = cels{i,1}.rain;
        end
    end
    y=max(rain)/3; %worst precipitation in the last 16 days
else
    flag_rain = 0;
    y=str2num(get(handles.precipitation,'String'));
end
```

8.2 Calculando altura de chuva de acordo com as coordenadas:

```
if ((latA <= 0) && (latA > -21))
    h_rain = 5;
elseif (latA <= -21)
    h_rain = 5 + 0.1*(abs(latA) + 21);
elseif ((latA > 0) && (lonA<-60 || ((lonA > -20) && (lonA < 60))))
    h_rain = 3.2 - 0.075*(abs(latA) - 35);
elseif (((latA > 0) && (latA <= 23)) && (((lonA >= -60) &&...
        (lonA <= -20)) || (lonA >= 60)))
    h_rain = 5;
elseif ((latA > 23) && (((lonA >= -60) && (lonA <= -20)) || (lonA >= 60)))
    h_rain = 5 - 0.075*(abs(latA) - 23);
end
```

8.3 Usando a tabela dos parâmetros experimentais

```
table = handles.table
test = abs(freq/(1e9)-table{:,1});
[~,idx]=min(test);
guidata(hObject,handles);
if strcmp(get(handles.polarization.SelectedObject,'String'),'Horizontal');
    k=table{idx,2};
    alfa=table{idx,3};
else
    k=table{idx,4};
    alfa=table{idx,5};
end
```

8.4 Cálculo final da atenuação

```
ang_rad = str2num(get(handles.elev_angle,'String'))*pi/180
Drain = (h_rain - height_tx/1000)/(sin(ang_rad)); % calculated in kms
yrain = k*power(y,alfa);

Lrain = Drain*yrain;
set(handles.output_rain,'String',Lrain);
handles.lrain = Lrain;
```

Published with MATLAB® R2015a

ANEXO 9 – *Script* para exportar os dados

9.1 Adequando os dados para exportar

```
aux = findobj('Style','edit')
cell_title = {'Technical Parameters'};
aux_1 = get(aux,'Tag');
aux_2 = get(aux,'String');
aux_2 = cellfun(@str2num,aux_2,'UniformOutput',false);
cell_aux = {'Parameters' 'Value'};
cell = [cell_aux;aux_1 aux_2];
```

9.2 Chamando o MATLAB Report Generator

```
assignin('base', 'cell', cell);
report gerar_report
```

Published with MATLAB® R2015a

ANEXO 10– Exemplo de exportação dos dados

Radio Link Budget Design

Export Data

Published 07-Dec-2015 22:39:35

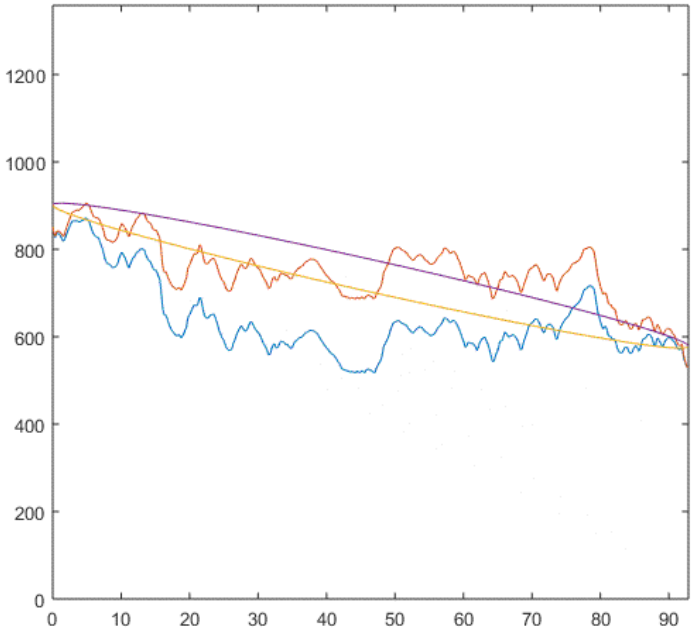
Contents

Chapter 1. Techincal Parameters Table

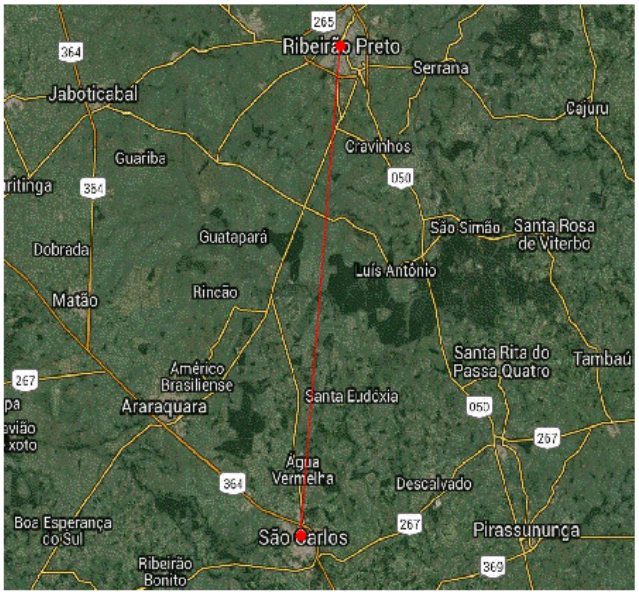
Parameters	Value
antenna_gain_rx	15
antenna_gain_tx	20
antenna_height_rx	50
antenna_height_tx	50
antenna_temp_rx	20
bandwidth	5000000
ber_calc	2.0750e-07
elev_angle	50
entrada_amostras	500
entrada_freq	5.0000e+09
entrada_latitudeA	-22.0071
entrada_latitudeB	-21.1719
entrada_longitudeA	-47.8944
entrada_longitudeB	-47.8122
link_margin	4
noise_figure	3
output_freespace	145.7840
output_rain	0.0585
power_tx	25
precipitation	6
sens_rx	-90

The table above resumes the tecnichal parameters which were tuned with the software.

Chapter 2. Terrain Information



Terrain Plots



Chapter 3. Link Considerations

This radio link has a total distance of 92.86kms
The antenna at point A is located at the altitude of 851.46m, placed 50.00m from the ground.
The antenna at point B is located at the altitude of 528.20m,placed 50.00m over the ground.
The receiver is currently performing with a 2.07502e-07 BER, receiving -89.84dBm of power. The modulation chosen was sixtyfour_qam.