



**Nota Técnica – Automatização do processamento digital de  
imagens do Satélite Amazônia-1 (WFI) para o Projeto  
DETER Amazônia.**

Thiago de Moraes Nishimura  
Luis Waldyr Rodrigues Sadeck  
Alessandra Rodrigues Gomes  
Nelson Cavalcante da Luz  
David Naoki Nishimura Kawamura

INPE  
São José dos Campos  
**2024**

**PUBLICADO POR:**

**Instituto Nacional de Pesquisas Espaciais - INPE**

**Gabinete do Diretor (GB)**

**Serviço de Informação e Documentação (SID)**

**Caixa Postal 515 - CEP 12.245-970**

**São José dos Campos - SP - Brasil**

**Tel.:(012)**

**Fax: (012)**

**E-mail:**

**CONSELHO DE EDITORAÇÃO:**

**Presidente:**

**Membros:**

**BIBLIOTECA DIGITAL:**

**REVISÃO E NORMALIZAÇÃO DOCUMENTÁRIA:**

**EDITORAÇÃO ELETRÔNICA:**

## LISTA DE FIGURAS

Figura 1: Fluxograma da automatização do Processamento Digital de Imagens.....	08
Figura 2: Obtenção da data do <i>script_amazonia1_txt.sh</i> .....	09
Figura 3: Obtenção da data do <i>script_amazonia1_txt_yesterday.sh</i> .....	09
Figura 4: <i>script_amazonia1_download.sh</i> .....	10
Figura 5: <i>script_amazonia1_move_u_w.sh</i> . .....	11
Figura 6: Script <i>AMAZONIA1_RGB.bat</i> . .....	12
Figura 7: Definição do ambiente e diretórios. ....	12
Figura 8: Geração da lista de cada órbita para a composição.....	13
Figura 9: Seleção das imagens .....	14
Figura 10: Leitura das bandas espectrais.....	14
Figura 11: Geração do gráfico de dispersão .....	15
Figura 12: Gráfico de dispersão e identificação das classes vegetação e solo para a imagem da órbita-ponto 035_015.....	15
Figura 13: Armazenamento dos endmembers em um arquivo .TXT .....	16
Figura 14: Normalização das frações .....	16
Figura 15: Obtenção dos endmembers a partir do arquivo .TXT .....	17
Figura 16: Função <i>mlme_2bands</i> .....	18
Figura 17: Função para processar todas as imagens disponíveis.....	19
Figura 18: Script <i>converter_8bits.py</i> .....	21
Figura 19: Script <i>contraste.py</i> .....	23
Figura 20: <i>script_amazonia1_move_imagens_w_u.sh</i> .....	25
Figura 21: Script de envio de mensagem.....	26
Figura 22: Definição da pasta de trabalho .....	27
Figura 23: Etapa de renomeação .....	27
Figura 24: Compressão dos arquivos.....	28
Figura 25: Criação da lista geoserver.txt.....	28
Figura 26: Envio dos arquivos via FTP .....	29
Figura 27: Crontab no servidor Linux .....	30
Figura 28: Script <i>executar.todos.scripts.sh</i> .....	31

## LISTA DE TABELAS

Tabela 1: Composição RGB manual X automatizada (órbita-ponto: 037_015). ....	34
Tabela 2: Fração solo manual X automatizada (órbita-ponto: 037_015). ....	35
Tabela 3: Fração vegetação manual X automatizada (órbita-ponto: 037_015).....	36
Tabela 4: Fração sombra manual X automatizada (órbita-ponto: 037_015). ....	37
Tabela 5: Fração solo com contraste manual X automatizada (órbita-ponto: 037_015).38	
Tabela 6: Fração vegetação com contraste manual X automatizada (órbita-ponto: 037_015).....	39
Tabela 7: Fração sombra com contraste manual X automatizada (órbita-ponto: 037_015).....	40

## SUMÁRIO

1 INTRODUÇÃO/HISTÓRICO DA DEMANDA .....	5
2 OBJETIVOS .....	7
3 METODOLOGIA .....	8
3.1 Processamento Digital de Imagens .....	9
3.1.1- Varredura, criação da lista de imagens e download .....	9
3.1.2- Exporte das imagens para o servidor Windows.....	10
3.1.3- Composição colorida.....	11
3.1.4- Endmembers.....	13
3.1.5- Modelo Linear de Mistura Espectral (MLME).....	16
3.1.6- Conversão das imagens para 8 bits.....	19
3.1.7- Contraste.....	22
3.1.8- Exporte das imagens para o servidor Linux .....	24
3.1.9- Envio de mensagem.....	26
3.1.10- Preparação de imagens para Web Map Service (WMS) e envio via FTP .....	26
3.2- Integração entre os Servidores .....	29
Figura 29 - Script executar_todos_scripts.sh .....	31
3.3- Validação.....	31
4 RESULTADOS.....	32
4.1- Desempenho e Eficiência: .....	32
4.2- Qualidade dos Resultados: .....	32
4.3- Exemplos de Saídas:.....	33
5 DESAFIOS E LIMITAÇÕES .....	41
6 CONCLUSÃO .....	41
7 AGRADECIMENTOS .....	42
8 REFERÊNCIAS .....	43

**Nota Técnica - Automatização do processamento digital de imagens do Satélite Amazônia-1 (WFI) para o Projeto DETER Amazônia.**

## **1 INTRODUÇÃO/HISTÓRICO DA DEMANDA**

O monitoramento ambiental, especialmente em regiões sensíveis como a região Amazônica, exige uma análise constante e precisa de imagens geradas por satélite. No entanto processamento manual dessas imagens, como o processo de composição de várias bandas espectrais em uma imagem RGB e a aplicação do Modelo Linear de Mistura Espectral (MLME), assim como dos Índices de vegetação e água, são uma rotina que consome muito tempo e está sujeita a erros humanos. Portanto a automatização desses processos surge como uma solução eficaz para otimizar o tratamento dessas imagens, aumentando assim a precisão do resultado, bem como garante uma consistência nas imagens geradas.

O satélite Amazônia-1 é um satélite totalmente brasileiro e foi lançado pelo Instituto Nacional de Pesquisas Espaciais (INPE) em 28 de fevereiro de 2021, ele desempenha um papel de extrema importância no monitoramento florestal da região amazônica, fornecendo imagens em média resolução e com alta taxa de revisita, que são utilizadas em diversas aplicações ambientais (INPE, 2021). As imagens geradas por este satélite são essenciais para o controle do desmatamento, queimadas e outros impactos ambientais na floresta (BOSCOLO et al., 2024). Com a automatização dos processos de baixar, composição das imagens, aplicação do MLME e índices, além de procedimentos de envio de mensagens, a equipe técnica consegue, de forma eficiente, maximizar a utilização dos dados gerados pelo Amazônia-1, auxiliando na obtenção de informações quase em tempo real dos dados de desflorestamento e degradação na Amazônia.

Diversas tecnologias e ferramentas foram essenciais para a automatização das rotinas de processamento digital de imagens de satélite no projeto DETER Amazônia. Dentre elas, destaca-se o uso do QGIS, que desempenhou um papel crucial na visualização, manipulação e análise das imagens, permitindo a verificação precisa dos resultados em cada etapa do fluxo automatizado.

A Geospatial Data Abstraction Library (GDAL), uma biblioteca open-source amplamente reconhecida no campo do processamento de dados geoespaciais, foi intensamente utilizada para leitura, escrita e manipulação de arquivos raster. Scripts em Python (.py) e batch (.bat) foram empregados para realizar composições de bandas espectrais em imagens RGB e aplicar o Modelo Linear de Mistura Espectral (MLME), assegurando a robustez dos procedimentos adotados.

A automatização foi implementada predominantemente em Python, devido à flexibilidade e à extensa biblioteca de pacotes científicos oferecidos pela linguagem, que têm demonstrado eficácia no processamento de grandes volumes de dados geoespaciais, conforme evidenciado em estudos recentes (LEMENKOVA e DEBEIR, 2022; SILVA et al., 2024). Para a gestão de dependências e ambientes, foi utilizado o Conda, que possibilitou a instalação controlada de pacotes como GDAL e numpy, em um ambiente específico configurado para este projeto, denominado gdal\_env.

A integração das ferramentas open-source foi facilitada pelo OSGEO4W, um conjunto abrangente de ferramentas geoespaciais voltado para ambientes Windows, que permitiu a execução de comandos e funções nativas do QGIS e GDAL diretamente por meio de scripts. Scripts batch (.bat) e shell (.sh) foram criados para automatizar tarefas rotineiras que, de outra forma, demandaria intervenção manual, como a configuração de ambientes Python, transferência de arquivos, organização de pastas e listas, composição de imagens e aplicação do MLME.

O armazenamento e a gestão dos processos automatizados foram centralizados em um servidor Ubuntu, que operou como a principal infraestrutura para o armazenamento das imagens processadas e garantiu a comunicação eficiente entre os ambientes Ubuntu e Windows por meio de scripts bash e protocolo SSH. Paralelamente, um servidor Windows foi dedicado ao processamento das imagens RGB e à aplicação do MLME, devido à compatibilidade com o OSGEO4W, sendo também o ambiente de execução dos scripts Python, assegurando a integridade e a eficiência do fluxo de trabalho automatizado.

A automatização da composição colorida das imagens foi desenvolvida para otimizar a tarefa de integração das imagens em bandas espectrais obtidas pelo satélite Amazônia-1 em uma única imagem RGB, facilitando o tratamento posterior dessas imagens. O processo consiste em seleção e combinação de imagens que possuam a mesma órbita

ponto e data, as quais são unidas tanto em ordem, quanto em bandas específicas, formando assim imagens prontas para serem analisadas.

**Processamento das bandas:** As imagens obtidas pelo satélite Amazônia-1 são fornecidas em múltiplas bandas espectrais, e estão disponíveis no site <http://cbers9.dpi.inpe.br:8089/files/AMAZONIA1/>, as quais representam diferentes comprimentos de ondas da luz que é refletida pela superfície da terra. O satélite Amazônia-1 possui 4 bandas, 3 bandas de frequência no espectro visível e uma banda no infravermelho próximo, sendo elas as bandas 1(Blue), 2(Green), 3(Red) e 4(NIR), nesse contexto o satélite apresenta três combinações de bandas:

- **Cor verdadeira:** Composição colorida RGB 3-2-1
- **Infravermelho colorido:** Composição colorida RGB 4-3-2
- **Análise da vegetação:** Composição colorida RGB 3-4-2

**Nível das imagens (L2 e L4):** As imagens do satélite Amazônia-1 são classificadas em vários níveis de processamento, como o nível 2 (L2), que corresponde às imagens com correção radiométrica e geométrica de sistema, e o nível 4 (L4), que são imagens ortoretificadas, ou seja, imagens com correção radiométrica e geométrica de sistema refinada pelo uso de pontos de controle e de um modelo digital de elevação de terreno. As imagens de nível L4 já estão prontas para uso, sem a necessidade de procedimentos adicionais. O script foi desenvolvido para priorizar as imagens de nível L4 sempre que possível e substituindo as de nível L2 sempre que possível.

## 2 OBJETIVOS

O principal objetivo dessa nota técnica é descrever os processos de automatização empregados no projeto DETER Amazônia dentro da COEAM, com isso, abordaremos os principais objetivos da automatização.

**Reducir o tempo de processamento das imagens:** Automatizar tarefas manuais e repetitivas, como a composição de imagens de várias bandas em uma imagem RGB e a aplicação do MLME, permitindo que um grande volume de dados seja analisado de forma otimizada e eficiente gastando um quinto do tempo utilizado pelo método manual.

**Minimizar erros humanos:** A automatização garante a consistência na execução dos processos, evitando que erros humanos sejam cometidos durante a geração dos dados, tais como erros de configuração e parametrização.

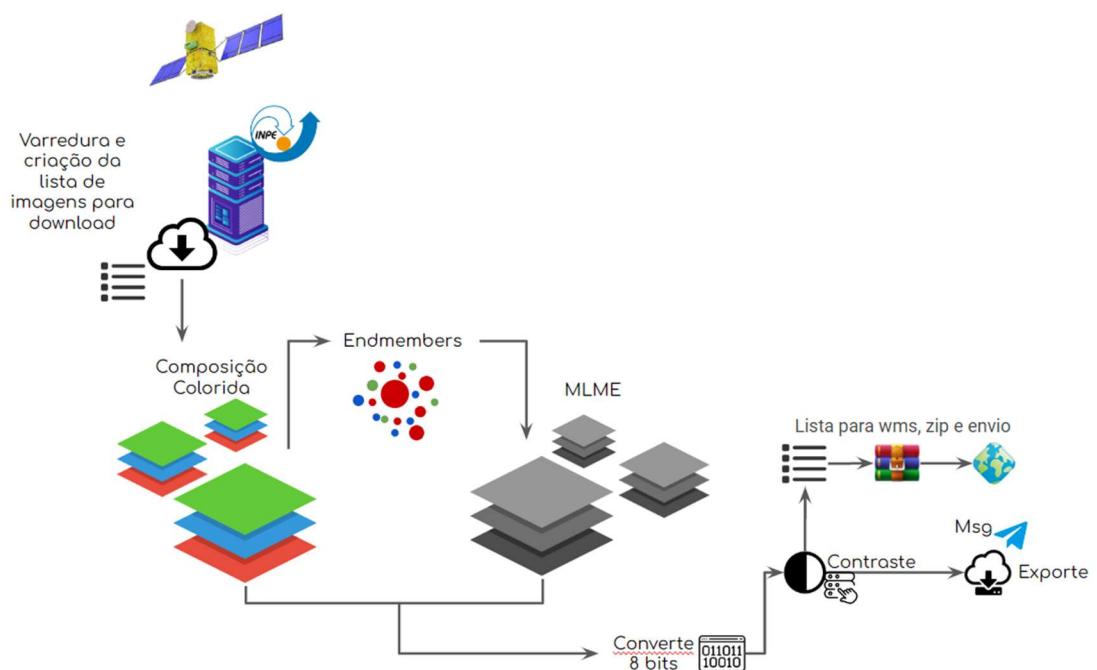
**Padronizar as imagens geradas:** A automatização da aplicação do MLME garante que as frações solo, vegetação e sombra sejam calculadas de forma padronizada, resultando em imagens padronizadas e comparáveis ao longo do tempo.

**Escalabilidade:** Facilitar a escalabilidade das tarefas, permitindo assim o processamento contínuo de imagens à medida que novos dados são fornecidos pelo satélite Amazônia-1.

### 3 METODOLOGIA

Para a implementação da automatização do Processamento Digital de Imagens, foram utilizadas várias ferramentas e tecnologias, as quais, em conjunto, permitiram a padronização e otimização das análises.

Figura 1 - Fluxograma da automatização do Processamento Digital de Imagens



### 3.1 Processamento Digital de Imagens

#### 3.1.1- Varredura, criação da lista de imagens e download

Essa primeira parte da metodologia é realizada em um Servidor Linux e foi desenvolvida com o objetivo de automatizar o processo de aquisição de imagens do satélite Amazônia-1 que estão armazenadas em um servidor FTP. A abordagem consiste em duas etapas principais: varredura do servidor FTP para a criação de uma lista de links listando as imagens disponíveis, e a realização do download das imagens utilizando a lista de links criada para o ambiente local.

O processo inicial consiste na varredura sistemática do servidor FTP, acessível por meio do endereço <http://cbers9.dpi.inpe.br:8089/files/AMAZONIA1/>, que armazena as imagens adquiridas pelo satélite Amazônia-1. As imagens são organizadas em diretórios hierárquicos baseados nas datas de aquisição, onde cada subdiretório corresponde a uma data específica de coleta.

A varredura é realizada por dois scripts automatizados, denominados *script\_amazonia1\_txt.sh* e *script\_amazonia1\_txt\_yesterday.sh*. Esses scripts percorrem recursivamente os diretórios do servidor FTP recursivamente, identificando e listando todos os arquivos de imagem disponíveis, tanto do dia atual quanto para o dia anterior. Esse processo ocorre em intervalos de 30 minutos, alternadamente, utilizando o comando -d “yesterday” para obter os arquivos do dia anterior.

Figura 2 - Obtenção da data do script\_amazonia1\_txt.sh

```
# Obtendo as datas atuais
ano_mes=$(date +"%Y_%m")
data_atual=$(date +"%Y_%m_%d")

# Construindo a URL base, para iniciar pela pasta do mês atual
url_base="http://cbers9.dpi.inpe.br:8089/files/AMAZONIA1/\${ano\_mes}/"
```

Figura 3 - Obtenção da data do script\_amazonia1\_txt\_yesterday.sh

```
# Obtendo as datas atuais
ano_mes=$(date -d "yesterday" +"%Y_%m")
data_atual=$(date -d "yesterday" +"%Y_%m_%d")

# Construindo a URL base, para iniciar pela pasta do mês atual
url_base="http://cbers9.dpi.inpe.br:8089/files/AMAZONIA1/\${ano\_mes}/"
```

Após a identificação dos arquivos relevantes, o script gera automaticamente uma lista contendo os links diretos para cada uma das imagens encontradas no servidor. Essa lista é armazenada em um arquivo de texto (.txt), cujo nome reflete a data de aquisição das imagens, facilitando a organização e o acesso aos dados.

Na segunda etapa do processo, o arquivo .txt gerado na etapa anterior é utilizado pelo *script\_amazonia1\_download.sh*, que lê sequencialmente cada um dos links e efetua o download das imagens utilizando o comando *wget*, uma ferramenta nativa de ambientes Linux para transferências via HTTP. Cada arquivo de imagem é transferido e armazenado localmente no servidor, garantindo a integridade dos dados e a continuidade do processamento automatizado.

Figura 4 - *script\_amazonia1\_download.sh*

```
# Desenvolvido por: Thiago Nisimura
# email: thiago.nisimura@inpe.br

#!/bin/bash

# Caminho do arquivo .txt gerado pelo script inicial
nome_arquivo="/Amazonia1/${(date +"%Y_%m_%d")}_AMAZONIA1.txt"

# Caminho da pasta onde serão salvos os arquivos baixados
caminho_da_pasta="/Amazonia1/Imagens"

# Criando a pasta de destino se não existir
mkdir -p "$caminho_da_pasta"

# Lê o arquivo linha por linha, baixa e salva os arquivos.
while IFS= read -r url; do
    echo "Baixando $url"
    wget -P "$caminho_da_pasta" "$url"
done < "$nome_arquivo"
```

### 3.1.2- Exporte das imagens para o servidor Windows.

Com as imagens baixadas e localizadas no servidor Linux, é necessária a transferência das imagens para o ambiente Windows, ambiente este configurado e ambientado para realizar o Processo Digital de Imagens (PDI).

O servidor Linux utilizado foi o Ubuntu Server, e nele foi realizado a configuração de montagem de um diretório remoto do servidor windows, que foi definido em */mnt/deter*, diretório esse configurado para realizar o procedimento de PDI.

Através do *script\_amazonia1\_move\_imagens\_u\_w.sh* as imagens localizadas no servidor Linux são movidas para o servidor windows, onde serão utilizadas nas etapas seguintes.

Figura 5 - *script\_amazonia1\_move\_imagens\_u\_w.sh*

```
#!/bin/bash

# Diretório de origem
ORIGEM="/Amazonia1/Imagens"

# Diretório de destino no Windows
DESTINO="/mnt/deter"

# Transferir arquivos .tif
mv $ORIGEM/*.tif $DESTINO
```

### 3.1.3- Composição colorida

Para a geração da composição colorida das imagens do satélite Amazônia-1, sensor WFI, foram utilizadas as bandas espectrais RED (banda 3), NIR (banda 4) e GREEN (banda 2).

A composição colorida foi aplicada utilizando imagens dos níveis L2 e L4, e foi separado em três scripts: *AMAZONIA1\_RGB.bat*, *AMAZONIA1\_RGB\_L2.bat* e *AMAZONIA1\_RGB\_L4.bat*.

O processo é coordenado pelo script *AMAZONIA1\_RGB.bat*, que é responsável por gerenciar a execução dos outros dois scripts. Esse script executa os outros dois de forma sequencial, sem a necessidade de intervenção manual, garantindo que cada nível de imagem L2 e L4 sejam executados.

Figura 6 - Script AMAZONIA1\_RGB.bat.

```
REM Verifica se os scripts L2 e L4 existem antes de executá-los

REM Executar o script para o Nível L2
if exist "C:\DETER\Scripts\AMAZONIA1_RGB_L2.bat" (
    ECHO Executando composicao de imagens Nível L2...
    call "C:\DETER\Scripts\AMAZONIA1_RGB_L2.bat"
    ECHO Composicao de imagens Nível L2 concluída.
) else (
    ECHO O script AMAZONIA1_RGB_L2.bat não foi encontrado!
)
ECHO.

REM Executar o script para o Nível L4
if exist "C:\DETER\Scripts\AMAZONIA1_RGB_L4.bat" (
    ECHO Executando composicao de imagens Nível L4...
    call "C:\DETER\Scripts\AMAZONIA1_RGB_L4.bat"
    ECHO Composicao de imagens Nível L4 concluída.
) else (
    ECHO O script AMAZONIA1_RGB_L4.bat não foi encontrado!
)
ECHO.

ECHO Todas as composicoes foram realizadas com sucesso!
```

Os scripts *AMAZONIA1\_RGB\_L2.bat* e *AMAZONIA1\_RGB\_L4.bat* são semelhantes em estrutura e fluxo de execução, diferindo-se apenas pelo tratamento das imagens, onde cada qual processa apenas as imagens do nível da qual foi direcionado. Iremos tomar como exemplo o script de nível L4 para demonstrar a metodologia a seguir.

O script começa definindo o caminho para o ambiente OSGeo4W, que é utilizado para acessar a ferramenta GDAL, como *gdalbuildvrt* e *gdal\_translate*. Depois são definidos os diretórios de trabalho.

Figura 7 - Definição do ambiente e diretórios

```
REM Define o caminho para o OSGeo4W.bat
set OSGEO4W_ROOT="C:\Program Files\QGIS 3.28.9"
call %OSGEO4W_ROOT%\bin\o4w_env.bat

REM Definir diretórios de trabalho
set INPUT_DIR="C:\Deter"
set RGB_DIR="%INPUT_DIR%\RGB"
set ORIGINAIS_DIR="%INPUT_DIR%\ORIGINAIS"
set OUTPUT_10BITS_DIR="%INPUT_DIR%\10BITS"
```

Em seguida, verifica a existência de arquivos do nível L4 e, caso encontre, move essas imagens para o diretório de trabalho. A partir dos arquivos movidos, o script gera listas de arquivos de acordo com suas órbitas-ponto, que no caso da região Amazônica varia de 014 a 018, e seguindo a ordem específica de banda: 3, 4 e 2.

Figura 8 - Geração da lista de cada órbita para a composição

```
For %%x in (AMAZONIA%data%*_L4*.tif) do move "%%x" "%RGB_DIR%"  
  
cd %RGB_DIR%  
  
REM Gera as listas que serão usadas na composição RGB, mantendo a ordem R(3) G(4) B(2)  
dir /b *_014_L4_BAND3.tif,*_014_L4_BAND4.tif,*_014_L4_BAND2.tif > obt014.txt  
dir /b *_015_L4_BAND3.tif,*_015_L4_BAND4.tif,*_015_L4_BAND2.tif > obt015.txt  
dir /b *_016_L4_BAND3.tif,*_016_L4_BAND4.tif,*_016_L4_BAND2.tif > obt016.txt  
dir /b *_017_L4_BAND3.tif,*_017_L4_BAND4.tif,*_017_L4_BAND2.tif > obt017.txt  
dir /b *_018_L4_BAND3.tif,*_018_L4_BAND4.tif,*_018_L4_BAND2.tif > obt018.txt
```

O script utiliza o comando *gdalbuildvrt* para criar arquivos virtuais (.vrt) que agrupam as bandas necessárias em um único arquivo RGB, ao mesmo tempo formatando o nome do arquivo para o padrão desejado.

As imagens virtuais são convertidas para o formato .tif com 10 bits utilizando o comando *gdal\_translate*, que também define o tipo de dados como UInt16, mas limitando os bits de 0 a 1023. Após o término do procedimento, as imagens já compostas e nomeadas de forma padronizada, são movidas para o diretório de 10 bits.

### 3.1.4- Endmembers

Os endmembers representam as assinaturas espectrais puras das classes de interesse, como solo, vegetação e sombra. Para determinar os endmembers, foi utilizado um script Python (dispersograma.py) que analisa as imagens de satélite em formato TIFF em 10 bits, extraíndo as bandas espectrais relevantes para realizar a análise, nesse caso foram utilizadas as bandas 3(Red) e 4(NIR).

O processo de identificação dos endmembers foi realizado seguindo as seguintes etapas: Seleção das imagens; Leitura das Bandas espectrais; Geração do gráfico de dispersão; Identificação das classes; Extração dos endmembers e Armazenamento dos resultados.

O script foi configurado para identificar automaticamente as imagens contidas no diretório de trabalho, garantindo que nenhuma imagem fique de fora da seleção.

Figura 9 - Seleção das imagens

```
# Encontrar o arquivo TIFF mais recente na pasta
def find_latest_tiff(directory):
    # Listar todos os arquivos na pasta que terminam com ".tif"
    tiff_files = [f for f in os.listdir(directory) if f.endswith('.tif')]

    if not tiff_files:
        raise FileNotFoundError("Nenhum arquivo TIFF encontrado na pasta.")

    # Ordenar os arquivos pelo tempo de modificação (o mais recente será o último)
    tiff_files.sort(key=lambda x: os.path.getmtime(os.path.join(directory, x)))

    # Retornar o caminho completo do arquivo mais recente
    return os.path.join(directory, tiff_files[-1])

# Obter o caminho da imagem TIFF mais recente
composite_image_path = find_latest_tiff(directory_path)
print(f"Carregando a imagem: {composite_image_path}")
```

Com a imagem selecionada, as bandas espectrais de interesse (Bandas 3 e 4) são selecionadas, lidas e armazenadas em um DataFrame, no caso utilizamos a biblioteca do Pandas. Desta forma é possível obter cada pixel da imagem de forma representativa, utilizando as bandas selecionadas, permitindo uma análise detalhada dos dados. Um outro passo muito importante para a consistência dos dados, são eliminadas todas as linhas onde pelo menos um dos dados está ausente.

Figura 10 - Leitura das bandas espectrais

```
# Criar DataFrame com as bandas
df = pd.DataFrame({
    'Banda3': banda3.flatten().astype(int),
    'Banda4': banda4.flatten().astype(int)
})

# Excluir todas as linhas onde há pelo menos um dado ausente
df = df.dropna()
```

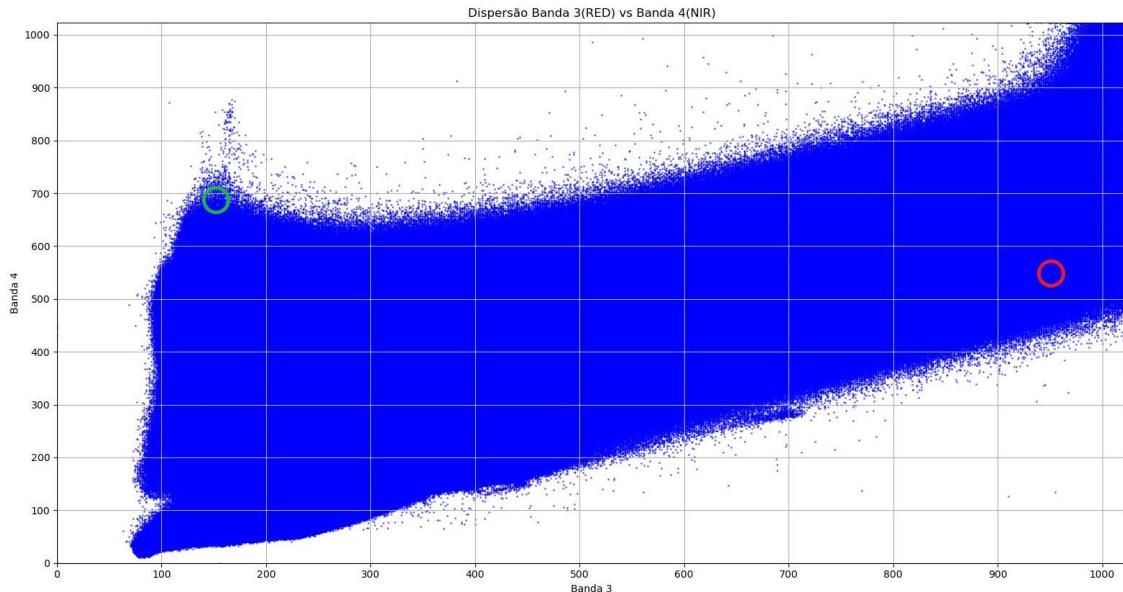
De posse desse DataFrame é gerado um gráfico de dispersão utilizando os valores da Bandas 3 representando o eixo X e a Banda 4 representando o eixo Y. Esse gráfico foi fundamental para visualizar a distribuição dos pixels e identificar os agrupamentos característicos das classes solo, vegetação e sombra.

Figura 11 - Geração do gráfico de dispersão

```
# Plotar os dados
plt.scatter(df['Banda3'], df['Banda4'], alpha=0.5, s=1, c='blue')
plt.xlabel('Banda 3')
plt.ylabel('Banda 4')
plt.title('Dispersão Banda 3(RED) vs Banda 4(NIR)')
plt.xlim(0, 1023)
plt.ylim(0, 1023)
plt.xticks(np.arange(0, 1024, 100))
plt.yticks(np.arange(0, 1024, 100))
plt.grid(True)
plt.show()
```

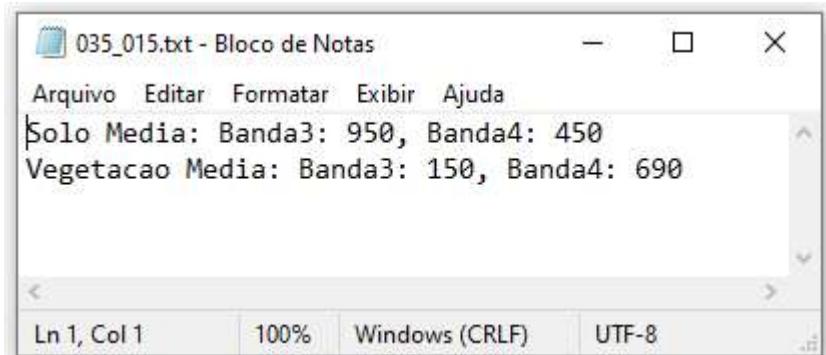
Com base no gráfico de dispersão que foi gerado, foi definido os valores endmembers para as classes (vegetação: círculo verde e solo: círculo vermelho) e salvando essas informações em um arquivo TXT, o qual foi gerado para cada órbita-ponto que representa o Bioma Amazônico, já os endmembers para o solo não foram coletados, pois após vários testes percebeu-se mais eficaz realizar o cálculo da inversa da vegetação, etapa essa realizada diretamente no script que realiza o MLME.

Figura 12 - Gráfico de dispersão e identificação das classes vegetação e solo para a imagem da órbita-ponto 035\_015



Após a definição das áreas correspondentes a cada classe, os pixels correspondentes foram isolados e suas médias espectrais, em ambas as bandas, foram calculadas. Esses valores médios representam os endmembers de cada classe, os quais são fundamentais para a aplicação do MLME.

Figura 13 - Armazenamento dos endmembers em um arquivo .txt



A identificação precisa dos endmembers é crucial para garantir a acurácia do modelo de mistura, permitindo uma melhor diferenciação entre as classes de cobertura do solo nas imagens de satélite. O script desenvolvido automatiza esse processo, assegurando que os endmembers sejam extraídos de maneira consistente e eficiente a partir das imagens mais recentes.

### 3.1.5- Modelo Linear de Mistura Espectral (MLME)

O MLME foi aplicado para decompor os pixels das imagens de satélite em frações que representam solo, vegetação e sombra. O MLME é utilizado para lidar com pixels mistos, especialmente em áreas de transição, como bordas de florestas e regiões agrícolas, onde diferentes materiais podem estar presentes em um único pixel.

O script *mlme.py* foi desenvolvido para aplicar o MLME em imagens .tif do satélite Amazônia-1, utilizando as bandas 3 e 4, para gerar três frações: solo, vegetação e sombra, normalizadas em 10 bits (0 a 1023). O script faz uso das bibliotecas numpy para manipulação de matrizes, gdal para leitura e escrita de dados de imagens e os e glob para busca de arquivos. Essas bibliotecas são utilizadas para realizar o processamento de imagens de satélite.

Figura 14 - Normalização das frações

```
# Função para normalizar as frações em 10 bits (0 a 1023)
def normalize_10bits(fraction):
    fraction = np.clip(fraction, 0, 1) # Mantém os valores no intervalo [0, 1]
    return (fraction * 1023).astype(np.uint16) # Converte para intervalo [0, 1023] e para uint16
```

Os endmembers são as assinaturas espectrais dos elementos que compõem a imagem. O script lê esses valores a partir do arquivo .txt gerado, analisando a imagem que está sendo processada no momento e obtendo o arquivo .txt correspondente à sua órbita-ponto.

Figura 15 - Obtenção dos endmembers a partir do arquivo .txt

```
# Função para ler os valores dos endmembers a partir do arquivo .txt
def read_endmember_values(txt_path):
    endmembers = []

    with open(txt_path, 'r') as file:
        lines = file.readlines()

        for line in lines:
            # A partir de cada linha, extraímos os valores das bandas 3 e 4
            if "Solo Media" in line:
                banda3 = int(line.split("Banda3:")[1].split(",")[0].strip())
                banda4 = int(line.split("Banda4:")[1].strip())
                endmembers.append([banda3, banda4])
            elif "Vegetacao Media" in line:
                banda3 = int(line.split("Banda3:")[1].split(",")[0].strip())
                banda4 = int(line.split("Banda4:")[1].strip())
                endmembers.append([banda3, banda4])

    if len(endmembers) != 2:
        raise ValueError(f"Valores de endmembers incorretos no arquivo {txt_path}")

    # Exibir os valores carregados para verificação
    print(f"Valores carregados de {txt_path}:")
    print(f"Solo: Banda3 = {endmembers[0][0]}, Banda4 = {endmembers[0][1]}")
    print(f"Vegetacao: Banda3 = {endmembers[1][0]}, Banda4 = {endmembers[1][1]}")

    return endmembers
```

A função *mlme\_2bands* lê as bandas 3 e 4 de cada imagens, empilhando-as para formar uma matriz de pixels. Com isso, é calculado as frações solo e vegetação, e utilizando os endmembers contidos nos arquivos .txt. As frações geradas são normalizadas e convertidas em imagens georreferenciadas no formato .tif

Diferente das frações vegetação e solo, a fração sombra não foi gerada a partir de endmembers presentes no arquivo .txt, foi gerado aplicando a inversa da fração vegetação. Aplicando essa técnica a geração da fração solo tornou-se mais legível.

Figura 16 - Função mlme\_2bands

```
# Função principal para realizar o MLME com 2 bandas
def mlme_2bands(image_path, endmember_values, normalization_func, bit_depth):
    # Abrir a imagem
    dataset = gdal.Open(image_path)

    if dataset is None:
        raise FileNotFoundError(f"Não foi possível abrir o arquivo {image_path}")

    # Ler as bandas 3 e 4
    banda3 = dataset.GetRasterBand(1).ReadAsArray() # Banda 1 representa a Banda 3
    banda4 = dataset.GetRasterBand(2).ReadAsArray() # Banda 2 representa a Banda 4

    # Obter dimensões
    rows, cols = banda3.shape

    # Espalhar as bandas em uma matriz
    pixels = np.stack((banda3.flatten(), banda4.flatten()), axis=-1).T

    # Matriz dos endmembers
    E = np.array(endmember_values).T

    # Calcular a inversa da matriz dos endmembers
    E_inv = np.linalg.pinv(E)

    # Calcular as frações de solo e vegetação
    fractions = np.dot(E_inv, pixels)

    # Normalizar as frações
    fractions = np.clip(fractions, 0, 1) # Clipping para garantir que não haja valores negativos
    fractions = normalization_func(fractions)

    # Retornar as frações de volta para o formato da imagem original
    fractions = fractions.T.reshape(rows, cols, -1)

    # Fração de vegetação e sombra como inversa
    solo = fractions[:, :, 0]
    vegetacao = fractions[:, :, 1]
    sombra = normalization_16bits(1 - (vegetacao / 1023)) # Inverso da fração de vegetação, normalizado

    # Nome das frações
    fraction_data = [
        "solo": solo,
        "vegetacao": vegetacao,
        "sombra": sombra
    ]

    # Salvar cada fração
    for fraction_name, fraction in fraction_data.items():
        # Salvar a fração como uma imagem de 8 bits
        driver = gdal.GetDriverByName('GTiff')

        # Criar o nome de saída para a fração
        output_path = f'{image_path.rsplit(".tif", 1)[0]}_{fraction_name}.tif'

        # Criar o arquivo de saída com tipo de dado 8-bit
        gdal_data_type = gdal.GDT_UInt16
        out_ds = driver.Create(output_path, cols, rows, 1, gdal_data_type)

        if out_ds is None:
            raise IOError(f"Erro ao criar o arquivo de saída para a fração {fraction_name}")

        # Escrever a fração na banda 1
        out_ds.GetRasterBand(1).WriteArray(fraction)

        # Definir as informações geo-referenciadas da imagem de saída
        out_ds.SetGeoTransform(dataset.GetGeoTransform())
        out_ds.SetProjection(dataset.GetProjection())

        # Fechar o dataset de saída
        out_ds = None

    # Fechar o dataset de entrada
    dataset = None
```

Para tornar o processo mais eficiente, o script é capaz de percorrer todas as imagens disponíveis em uma pasta específica, aplicando o MLME em cada uma delas. Cada imagem .tif recebe um arquivo .txt específico que contém os valores endmembers correspondentes à sua órbita-ponto. Isso permite uma análise detalhada e adequada a diferentes regiões, já que cada imagem pode ter características únicas devido à variação espacial.

Figura 17 - Função para processar todas as imagens disponíveis

```
# Função principal para processar todas as imagens na pasta 10BITS
def process_all_images_10bits(directory_10bits):
    tif_files = glob.glob(os.path.join(directory_10bits, "*.tif"))

    for image_path in tif_files:
        # Para cada imagem, buscamos o arquivo txt correspondente
        orbit_point = "_".join(image_path.split("_")[3:5]) # Extrai órbita-ponto do nome da imagem
        txt_file = os.path.join(directory_mlme_txt, f"{orbit_point}.txt")

        if os.path.exists(txt_file):
            print(f"Processando {image_path}...")
            print(f"Parâmetros de {txt_file}...")
            try:
                endmember_values = read_endmember_values(txt_file)
                mlme_2bands(image_path, endmember_values, normalize_10bits, 10)
                print(f"Concluído para {image_path}")
            except Exception as e:
                print(f"Erro ao processar {image_path}: {e}")
        else:
            print(f"Arquivo .txt correspondente não encontrado para {image_path}")

# Função para ler os valores endmembers de um arquivo .txt
def read_endmember_values(txt_file):
    with open(txt_file, 'r') as file:
        lines = file.readlines()
        endmembers = [float(line.strip()) for line in lines]
    return endmembers
```

A implementação do MLME com duas bandas é uma abordagem eficiente para a decomposição de pixels mistos em componentes individuais, permitindo uma análise mais precisa da cobertura do solo e dos impactos ambientais. A automatização do processo, possibilitada pelo script *mlme.py*, é particularmente vantajosa para a análise em larga escala e contínua, como a realizada no monitoramento do Bioma Amazônia pelo projeto do DETER Amazônia.

### 3.1.6- Conversão das imagens para 8 bits

A conversão das imagens de 10 bits para 8 bits é uma etapa essencial para padronizar o formato das imagens, facilitando sua visualização, análise e utilização em softwares que não suportam a profundidade de 10 bits. A redução de profundidades de bits visa otimizar o armazenamento e o processamento, tornando os dados mais compatíveis com uma maior variedade de ferramentas e requisitos de hardware.

As imagens geradas no processamento de MLME são originalmente produzidas em 10 bits, correspondendo a um intervalo de valores entre 0 e 1023. No entanto, muitos

sistemas de visualização e processamento de imagens trabalham de forma otimizada com profundidade de 8 bits, o que corresponde ao intervalo de valores entre 0 e 255.

O script responsável por essa etapa foi chamado de *converter\_8bits.py*, o qual utiliza as bibliotecas osgeo (GDAL) e Numpy, para realizar o acesso as imagens contidas no diretório e para realização das etapas de conversão em 8 bits.

O script segue o seguinte fluxo de processamento: A primeira etapa consiste em definir os diretórios de trabalho, definindo onde as imagens em 10 bits estão localizadas e onde as imagens de 8 bits serão salvas após a conversão; Após a definição dos respectivos diretórios, é realizada a etapa de conversão em 8 bits, onde é realizada a leitura da imagem original em 10 bits. No caso da imagem possuir 3 bandas (RGB), cada banda é lida e normalizada individualmente. No caso da imagem possuir apenas uma única banda, no caso das frações solo, sombra e vegetação, o processo de normalização é similar, mas sendo necessário a normalização de apenas uma banda.

Para manter a integridade espacial dos dados, a nova imagem é configurada para preservar as informações geográficas da imagem original, incluindo a projeção e transformação geográfica. Por fim o script processa todas as imagens contidas no diretório 10 bits, realizando a conversão e salvando os arquivos resultantes na pasta 8 bits com uma nomenclatura que reflete a mudança.

Figura 18 - Script converter\_8bits.py

```
import os
import numpy as np
from osgeo import gdal

# Caminhos das pastas
pasta_10bits = r'C:\DETER\10BITS'
pasta_8bits = r'C:\DETER\8BITS'

# Criar a pasta 8BITS se não existir
if not os.path.exists(pasta_8bits):
    os.makedirs(pasta_8bits)

# Função para converter de 10 bits para 8 bits
def converter_para_8bits(caminho_entrada, caminho_saida):
    # Abrir a imagem original
    dataset = gdal.Open(caminho_entrada)

    # Verificar o número de bandas
    num_bandas = dataset.RasterCount

    # Se for uma imagem RGB (com 3 bandas)
    if num_bandas == 3:
        # Criar uma nova imagem 8 bits com 3 bandas
        driver = gdal.GetDriverByName('GTiff')
        nova_imagem = driver.Create(caminho_saida, dataset.RasterXSize, dataset.RasterYSize, 3, gdal.GDT_Byte)

        # Para cada banda, realizar a normalização e salvar
        for i in range(1, num_bandas + 1):
            banda = dataset.GetRasterBand(i)
            dados = banda.ReadAsArray()

            # Normalizar de 10 bits para 8 bits (0-1023 para 0-255)
            dados_8bits = np.clip((dados / 1023.0) * 255, 0, 255).astype(np.uint8)

            # Escrever os dados normalizados na nova banda
            nova_imagem.GetRasterBand(i).WriteArray(dados_8bits)

    # Se for uma imagem de banda única (como solo, sombra, vegetação)
    else:
        banda = dataset.GetRasterBand(1)
        dados = banda.ReadAsArray()

        # Normalizar de 10 bits para 8 bits (0-1023 para 0-255)
        dados_8bits = np.clip((dados / 1023.0) * 255, 0, 255).astype(np.uint8)

        # Criar uma nova imagem 8 bits com uma banda
        driver = gdal.GetDriverByName('GTiff')
        nova_imagem = driver.Create(caminho_saida, dataset.RasterXSize, dataset.RasterYSize, 1, gdal.GDT_Byte)
        nova_imagem.GetRasterBand(1).WriteArray(dados_8bits)

    # Manter a geoinformação da imagem original
    nova_imagem.SetGeoTransform(dataset.GetGeoTransform())
    nova_imagem.SetProjection(dataset.GetProjection())

    # Fechar os datasets
    dataset = None
    nova_imagem = None

# Percorrer todos os arquivos na pasta 10BITS
for arquivo in os.listdir(pasta_10bits):
    if arquivo.endswith('.tif'):
        caminho_entrada = os.path.join(pasta_10bits, arquivo)

        # Substituir "10bits" por "8bits" no nome do arquivo
        novo_nome = arquivo.replace('10bits', '8bits')
        caminho_saida = os.path.join(pasta_8bits, novo_nome)

        # Converter a imagem para 8 bits e salvar na pasta 8BITS
        converter_para_8bits(caminho_entrada, caminho_saida)
        print(f"Imagen convertida e renomeada: {novo_nome}")

print("Conversão e renomeação concluídas!")
```

### **3.1.7- Contraste**

A aplicação de contraste foi realizada nas imagens de frações solo, sombra e vegetação que estão no diretório de 8 bits, não sendo aplicada ao diretório de 10 bits, pois essa etapa de contraste foi desenvolvida voltada a equipe que realiza o monitoramento do bioma Amazônia, equipe essa que utiliza as imagens em 8 bits. Esse procedimento tem como objetivo melhorar a visibilidade dos detalhes presentes nas imagens, aumentando o contraste entre as diferentes regiões, facilitando a interpretação visual e análises subsequentes.

O contraste foi ajustado utilizando a técnica de percentile stretch, que consiste em aplicar uma transformação linear baseada em percentis. Para a fração de solo, foi utilizado um stretching com percentis de 12% e 88%, enquanto para as frações de sombra e vegetação, foram utilizados percentis de 2% a 98%. Esse ajuste permite destacar as características específicas de cada fração, melhorando a discriminação entre as classes.

O script *contraste.py* desenvolvido para esta etapa, realiza as seguinte etapas:

**I- Leitura da imagem de fração:** As imagens de fração solo, sombra e vegetação são carregadas na memória para processamento.

**II- Cálculo dos percentis e aplicação do Stretching:** Para cada imagem, os valores de corte inferiores e superiores são definidos de acordo com os percentis especificados. Em seguida é aplicada uma normalização para ajustar os valores dos pixels dentro da faixa de 0 a 255, correspondente a 8 bits de profundidade.

**III- Processamento das frações no diretório:** O script percorre automaticamente a pasta contendo as imagens de 10 bits, identificando as frações de solo, sombra e vegetação para aplicar o contraste apropriado de acordo com a faixa de percentis definida para cada tipo de fração.

**IV- Geração da imagem com contraste aplicado:** Após o ajuste do contraste, uma nova imagem é gerada e salva, mantendo as informações geoespaciais da imagem original. O nome da imagem resultante inclui o sufixo “\_contraste”, indicando que o processamento de contraste foi realizado.

Figura 19 - Script contraste.py

```
import numpy as np
from osgeo import gdal
import os
import glob

# Função para aplicar stretching percentilico no intervalo de 0 a 255
def apply_percentile_stretch(image_array, low_percentile, high_percentile):
    low_value = np.percentile(image_array, low_percentile)
    high_value = np.percentile(image_array, high_percentile)

    # Evitar divisão por zero caso todos os valores sejam iguais
    if low_value == high_value:
        return image_array # Retorna a mesma imagem se não houver variação

    # Aplicar transformação Percentilica para a faixa de 0 a 255
    stretched_image = np.clip((255 * (image_array - low_value)) / (high_value - low_value)), 0, 255).astype(np.uint16)

    return stretched_image

# Função para aplicar o stretching percentilico à uma fração e salvar a nova imagem
def process_contrast_fraction(image_path, low_percentile, high_percentile):
    # Abrir a imagem da fração
    dataset = gdal.Open(image_path)

    if dataset is None:
        raise FileNotFoundError(f"Não foi possível abrir o arquivo {image_path}")

    # Ler a banda da fração
    fraction_band = dataset.GetRasterBand(1).ReadAsArray()

    # Aplicar stretching percentilico
    contrasted_fraction = apply_percentile_stretch(fraction_band, low_percentile, high_percentile)

    # Salvar a imagem processada com o sufixo "_contraste"
    driver = gdal.GetDriverByName('GTiff')
    output_path = f'{image_path.rsplit(".tif", 1)[0]}_contraste.tif'

    out_ds = driver.Create(output_path, dataset.RasterXSize, dataset.RasterYSize, 1, gdal.GDT_UInt16)
    out_ds.GetRasterBand(1).WriteArray(contrasted_fraction)

    # Definir as informações geo-referenciadas da imagem de saída
    out_ds.SetGeoTransform(dataset.GetGeoTransform())
    out_ds.SetProjection(dataset.GetProjection())

    # Fechar os datasets
    out_ds = None
    dataset = None

    print(f"Contraste aplicado e imagem salva em {output_path}")

# Função para processar todas as frações na pasta
def process_fractions_with_contrast(directory_8bits):
    # Padrão para as frações (solo, sombra, vegetação)
    fraction_files = glob.glob(os.path.join(directory_8bits, "*_solo.tif")) + \
                     glob.glob(os.path.join(directory_8bits, "*_sombra.tif")) + \
                     glob.glob(os.path.join(directory_8bits, "*_vegetacao.tif"))

    for fraction_path in fraction_files:
        if "_solo.tif" in fraction_path:
            print(f"Processando contraste para {fraction_path} com percentis 12 e 88...")
            process_contrast_fraction(fraction_path, 12, 88)
        elif "_sombra.tif" in fraction_path or "_vegetacao.tif" in fraction_path:
            print(f"Processando contraste para {fraction_path} com percentis 2 e 98...")
            process_contrast_fraction(fraction_path, 2, 98)

    # Exemplo de uso
directory_8bits = 'C:/DETER/8BITS'

# Aplicar contraste percentilico nas frações solo, sombra e vegetação
process_fractions_with_contrast(directory_8bits)
```

### **3.1.8- Exporte das imagens para o servidor Linux**

Após o término da execução dos procedimentos de PDI, as imagens estarão prontas para serem utilizadas pelos intérpretes. Neste momento é executado de forma automática o *script\_amazonial\_move\_imagens\_w\_u.sh* o qual move as imagens do servidor windows para o servidor Linux, separando as imagens processadas em diretórios organizados em 8 bits e 10 bits, e dentro de cada uma dessas pastas as imagens são organizadas em datas de obtenção das imagens pelo satélite.

Figura 20 - script\_amazonia1\_move\_imagens\_w\_u.sh

```
# Definir diretórios
SOURCE_DIR_8BITS=/mnt/deter/8BITS/
DEST_DIR_8BITS=/Amazonia1/Imagens/8BITS/
SOURCE_DIR_10BITS=/mnt/deter/10BITS/
DEST_DIR_10BITS=/Amazonia1/Imagens/10BITS/
DISCORD_SCRIPT_PATH=/scripts/mensagem.sh

# Função para mover arquivos
move_files() {
    local SOURCE_DIR=$1
    local DEST_DIR=$2
    for file in "$SOURCE_DIR"/*; do
        # Extrair data do nome do arquivo
        filename=$(basename "$file")
        date=$(echo "$filename" | grep -oP '\d{8}')

        # Criar diretório de destino se não existir
        target_dir="$DEST_DIR$date"
        if [ ! -d "$target_dir" ]; then
            mkdir -p "$target_dir"
            new_folder=true
        else
            new_folder=false
        fi

        # Mover arquivo para o diretório de destino
        mv "$file" "$target_dir/"

        # Verificar se a operação foi bem-sucedida
        if [ $? -eq 0 ]; then
            echo "Arquivo $filename movido com sucesso para $target_dir em $(date)"
            files_moved=true
        else
            echo "Falha ao mover o arquivo $filename em $(date)"
            files_moved=false
        fi
    done
}

# Mover imagens 10 bits
move_files "$SOURCE_DIR_10BITS" "$DEST_DIR_10BITS"

# Mover imagens 8 bits
move_files "$SOURCE_DIR_8BITS" "$DEST_DIR_8BITS"

# Acionar o script de envio de mensagem ao Discord, se necessário
if [ "$new_folder" = true ] || [ "$files_moved" = true ]; then
    echo "Novas imagens foram movidas para o servidor. Acionando script de notificação..."
    bash "$DISCORD_SCRIPT_PATH" "$target_dir"
else
    echo "Nenhuma nova imagem foi adicionada. Script de notificação não acionado."
fi
```

O processo de PDI requer um grande uso de memória RAM, memória virtual e processador, motivo esse que, durante o processamento das imagens, torna o servidor em estado de “lentidão”, este foi o principal motivo para realizar essa processo de automatização em servidores separados, onde o servidor Linux realiza a varredura, download e armazenamento das imagens e o servidor Windows realizando apenas o PDI.

### 3.1.9- Envio de mensagem

Ao mover todas as imagens para o servidor Linux, onde as imagem ficam armazenadas e disponíveis para acesso, é enviado uma mensagem para o grupo da equipe de

monitoramento, através do script *mensagem.sh* para o programa Discord, onde é informado que há imagens disponíveis para serem analisadas, evitando que a equipe a todo momento acessando o diretório para verificar se há imagens disponíveis.

Figura 21 - Script de envio de mensagem

```
# URL do Webhook do Discord
WEBHOOK_URL="https://discord.com/api/webhooks/CHAVE-TOKEN"

# Pasta que recebeu as novas imagens (passada como argumento)
PASTA="$1"

# Lista de mensagens animadoras
MENSAGENS=(
    "Vamos com tudo, equipe! Mais um dia de sucesso!"
    "Ótimo trabalho, pessoal! Continuem assim!"
    "Aqui estão as imagens. Agora respirem fundo e sigam em frente."
    "# Demais modelos de imagens...
)

# Selecionar uma mensagem aleatória
MENSAGEM_ALEATORIA=${MENSAGENS[$RANDOM % ${#MENSAGENS[@]}]}

# Criar a mensagem personalizada em JSON
MESSAGE=$(cat <<EOF
{
    "content": null,
    "embeds": [
        {
            "title": "Olá equipe DETER Amazônia, temos imagens novas!",
            "description": "\nPasta: $PASTA\n\n$MENSAGEM_ALEATORIA\n\nINPE - Instituto Nacional de Pesquisas Espaciais\nCOEAM - Coordenação Espacial da Amazônia",
            "color": 65280
        }
    ]
}
EOF
)

# Enviar notificação para o Discord
curl -H "Content-Type: application/json" \
-X POST \
-d "$MESSAGE" \
$WEBHOOK_URL
```

### 3.1.10- Preparação de imagens para Web Map Service (WMS) e envio via FTP

Após as etapas de processamento das imagens, incluindo composição colorida, aplicação do Modelo Linear de Mistura Espectral (MLME), conversão de bits, aplicação de contraste, análise e polígonos de alertas gerados pela equipe do DETER Amazônia, é necessário preparar as imagens que foram utilizadas para gerar esses alertas, para serem enviadas ao Web Map Service (WMS). Essas imagens são enviadas para validação por meio de um servidor FTP.

Para automatizar esse processo foi desenvolvido um script em lote, chamado *enviar\_geoserver\_ftp.bat*, que realiza as seguintes tarefas:

**I- Definição da pasta de trabalho:** A primeira etapa de execução do script é definir a pasta de trabalho onde os arquivos a serem tratados serão alocados.

Figura 22 - Definição da pasta de trabalho

```
:: Define o caminho da pasta
set "pasta_padrao=C:\DETER\ENVIO_FTP"

:: Navega para a pasta
cd /d "%pasta_padrao%"
```

**II- Renomeação dos arquivos:** Com os diretórios definidos, o script segue para a etapa de renomeação dos arquivos. Essa etapa é importante para evitar que arquivos sejam enviados fora do padrão de nomeação já definido, podendo ocasionar erros ao serem enviados para o servidor WMS.

Por padrão é removido os níveis de L2 e L4 contidos no nome dos arquivos baixados, bem como substituir, no caso do satélite Amazônia-1, “AMAZONIA\_1” POR “AMAZONIA-1”, o script varre a pasta onde os arquivos foram alocados realizando essa renomeação para cada arquivo da pasta.

Figura 23 - Etapa de renomeação

```
:: Renomeia os arquivos .tif
for %%f in (*.tif) do (
    set "nome_arquivo=%%~nf"

    :: Remove os níveis L2 e L4 do nome dos arquivos
    set "nome_arquivo=!nome_arquivo:_L2=!"
    set "nome_arquivo=!nome_arquivo:_L4=!"

    :: Renomeia as imagens do Amazonia-1
    set "nome_arquivo=!nome_arquivo:AMAZONIA_1=AMAZONIA-1!"

    :: Renomeia as imagens do CBERS-4
    set "nome_arquivo=!nome_arquivo:CBERS_4=CBERS-4!"

    :: Renomeia as imagens do CBERS-4A
    set "nome_arquivo=!nome_arquivo:CBERS_4A=CBERS-4A!"

    ren "%%f" "!nome_arquivo!.tif"
)
```

**III- Compressão dos arquivos:** A compressão dos arquivos é realizada comprimindo cada uma das imagens que estão no formato .tif para um arquivo .zip e não criando apenas um arquivo comprimido contendo todas as imagens. Após comprimir todas as imagens

para um formato .zip, as imagens .tif são deletadas da pasta, desse modo, deixando na pasta apenas arquivos .zip.

Figura 24 - compressão dos arquivos

```
:: Comprimi cada arquivo .tif em um arquivo .zip separado
for %%f in (*.tif) do (
    PowerShell Compress-Archive -Path "%%f" -DestinationPath "%~nf.zip"
    del "%%f"
)
```

**IV- Criação da lista de arquivos para o WMS:** Com os arquivos comprimidos, é hora de gerar o arquivo geoserver.txt que irá conter a lista com o nome de cada arquivo que estará sendo enviado por FTP, adicionando um ponto e vírgula ao final de cada linha.

Antes criar a lista e salvar dentro do arquivo geoserver.txt, o script garante que esse arquivo esteja ausente para não haver resíduos de outra lista salva anteriormente, portanto o script verifica se há um arquivo geoserver.txt na pasta, caso haja, ele deleta e cria um novo vazio. Com a garantia de que o arquivo está vazio, é realizada a etapa de criação da lista.

Figura 25 - Criação da lista geoserver.txt

```
:: Define o nome do arquivo de saída
set "saida_arquivo=%pasta_padrao%\geoserver.txt"

:: Verifica se o arquivo de saída já existe na pasta e o deleta
if exist "%saida_arquivo%" del "%saida_arquivo%"

:: Lista todos os arquivos .zip na pasta e escreve no arquivo de saída com um ";" no final
for /r "%pasta_padrao%" %%f in (*.zip) do (
    echo "%~nxz;" >> "%saida_arquivo%"
)
```

**V- Envio dos arquivos via FTP:** Com os arquivos renomeados para o padrão, comprimidos e gerada a lista geoserver.txt, os arquivos já estão prontos para serem enviados via FTP para o servidor WMS para validação.

O script gera um arquivo *ftp.txt* temporário contendo todas as informações necessárias para acessar o servidor FTP (IP do servidor FTP, usuário e senha) e realizar o envio das arquivos .zip e do arquivo *geoserver.txt*. Após o envio de todos os arquivos, esse arquivo temporário é deletado.

Foi tomado o devido cuidado para o script realizar o envio do arquivo .txt, somente após o envio de todos os arquivos .zip, pois no servidor WMS há uma rotina de varredura de tempo em tempo em busca do arquivo *geoserver.txt*, onde ele coleta esse arquivo e analiza se todas as imagens dessa lista estão presentes, caso essa varredura capture esse

arquivo antes de todos os arquivos .zip serem enviado, irá ocasionar um erro, portanto, para evitar essa falha, tomou-se essa cautela.

Figura 26 - Envio dos arquivos via FTP

```
:: Configurações FTP
set "ftp_server=IP_DO_SERVIDOR_FTP"
set "ftp_user=USUARIO_FTP"
set "ftp_password=SENHA_ACESSO"

:: Cria um arquivo de script FTP
(
echo open %ftp_server%
echo user %ftp_user%
echo %ftp_password%
echo binary
echo cd /DETER/AMAZONIA
for %%f in (%pasta_padrao%\*.zip) do echo put %%f
echo put %pasta_padrao%\geoserver.txt
echo quit
) > %pasta_padrao%\ftp.txt

:: Executa o script FTP
ftp -n -s:%pasta_padrao%\ftp.txt

:: Limpa o arquivo de script FTP temporário
del %pasta_padrao%\ftp.txt
```

### 3.2- Integração entre os Servidores

Para o processo de automatização do PDI funcionar de forma mais otimizada, tornou-se necessário a utilização de dois servidores funcionando em sincronia, um servidor Linux para realizar as tarefas de varredura, criação das listas dos links, download das imagens e gerenciamento do fluxo do processamento através de scripts e um servidor Windows para realizar todo a etapa do PDI.

A primeira configuração realizada para integrar os servidores foi mapear uma pasta no servidor Linux para acessar a pasta onde é realizado o PDI das imagens no Servidor Windows, com esse mapeamento estruturado permitiu a comunicação e transferências entre os ambientes mais fluidos e otimizados.

No servidor windows foi realizado a configuração e instalação do QGis 3.28.9, versão essa a escolhida, pois é a mais utilizada pela equipe e segundo testes a versão que menos apresenta falhas e erros. Outro software que foi necessário a instalação e configuração foi o Anaconda, uma ferramenta que integra linguagens e softwares para ciência de dados e programação científica, ela oferece um conjunto de ferramentas e pacotes pré-instalados, como o Numpy, Pandas e Matplotlib.

O servidor Linux foi escolhido para gerenciar o fluxo de processamento pois ele possui um gerenciador nativo de agendamentos de comandos chamado Crontab. O crontab é

disponibilizado em formato de um arquivo, onde é possível informar os dias da semana, hora e o comando a ser executado, podendo gerar assim uma sequência de comandos a serem executados durante o dia de forma automática.

Figura 27 - Crontab no servidor Linux

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# For details see man 4 crontabs

# Example of job definition:
# ----- minute (0 - 59)
# | ----- hour (0 - 23)
# | | ----- day of month (1 - 31)
# | | | ----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# * * * * * user-name command to be executed

#####
##### Executar Todos os Scripts #####
#####

#00 05 * * * /scripts/executar_todos_scripts_yesterday.sh
#00 06 * * * /scripts/executar_todos_scripts.sh
#30 06 * * * /scripts/executar_todos_scripts_yesterday.sh
00 07 * * * /scripts/executar_todos_scripts.sh
30 07 * * * /scripts/executar_todos_scripts_yesterday.sh
00 08 * * * /scripts/executar_todos_scripts.sh
30 08 * * * /scripts/executar_todos_scripts_yesterday.sh
00 09 * * * /scripts/executar_todos_scripts.sh
30 09 * * * /scripts/executar_todos_scripts_yesterday.sh
00 10 * * * /scripts/executar_todos_scripts.sh
30 10 * * * /scripts/executar_todos_scripts_yesterday.sh
00 11 * * * /scripts/executar_todos_scripts.sh
30 11 * * * /scripts/executar_todos_scripts_yesterday.sh
00 12 * * * /scripts/executar_todos_scripts.sh
30 12 * * * /scripts/executar_todos_scripts_yesterday.sh
00 13 * * * /scripts/executar_todos_scripts.sh
30 13 * * * /scripts/executar_todos_scripts_yesterday.sh
00 14 * * * /scripts/executar_todos_scripts.sh
30 14 * * * /scripts/executar_todos_scripts_yesterday.sh
00 15 * * * /scripts/executar_todos_scripts.sh
30 15 * * * /scripts/executar_todos_scripts_yesterday.sh
00 16 * * * /scripts/executar_todos_scripts.sh
30 16 * * * /scripts/executar_todos_scripts_yesterday.sh
00 17 * * * /scripts/executar_todos_scripts.sh
30 17 * * * /scripts/executar_todos_scripts_yesterday.sh
00 18 * * * /scripts/executar_todos_scripts.sh
30 18 * * * /scripts/executar_todos_scripts_yesterday.sh
00 19 * * * /scripts/executar_todos_scripts.sh
#00 20 * * * /scripts/executar_todos_scripts.sh
#00 21 * * * /scripts/executar_todos_scripts.sh
#00 22 * * * /scripts/executar_todos_scripts.sh
#00 23 * * * /scripts/executar_todos_scripts.sh
```

No crontab foi configurado a chamada de apenas dois scripts, um executando a rotina para as imagens do dia corrente e outro para executar a rotina para as imagens do dia anterior. Cada um desses scripts foram configurados para chamar os outros scripts de forma sequencial, onde o próximo só é acionado após o término do anterior. Eles são acionados a cada 30 minutos, de forma alternada entre eles.

Figura 28 - Script executar\_todos\_scripts.sh

```
#!/bin/bash

# Executar obtenção dos links do satelite Amazonia1
./scripts/script_amazonia1_txt.sh &&

# Executar download das imagens
./scripts/script_amazonia1_download.sh &&

# Mover as imagens
./scripts/script_amazonia1_move_imagens_u_w.sh &&

# Composição das imagens
sshpass -p 'senha_acesso' ssh usuario_acesso@Coeam-server 'cmd /c "C:\DETER\scripts\AMAZONIA_RGB.bat"' &&

# Modelo Linear de Mistura Espectral
sshpass -p 'senha_acesso' ssh usuario_acesso@Coeam-server 'cmd /c "C:\DETER\scripts\mlme.bat"' &&

# Realiza o Contraste
sshpass -p 'senha_acesso' ssh usuario_acesso@Coeam-server 'cmd /c "C:\DETER\scripts\contraste.bat"' &&

# Converter as imagens para 8 bits
sshpass -p 'senha_acesso' ssh usuario_acesso@Coeam-server 'cmd /c "C:\DETER\scripts\converter_8bits.bat"' &&

# Mover imagens compostas
./scripts/script_amazonia1_move_imagens_w_u.sh

# Verificação de sucesso
if [ $? -eq 0 ]; then
    echo "Todos os scripts foram executados com sucesso."
else
    echo "Houve um erro durante a execução dos scripts."
fi
```

### 3.3- Validação

Para realizar a validação das imagens geradas pela automatização, foi realizado o teste A/B, também chamado de teste de divisão ou teste de bucket. Esse método consiste em comparar uma versão de controle (A) com uma versão de controle (B), para medir qual delas apresenta melhor desempenho para o critério avaliado.

O teste foi aplicado à equipe do DETER Amazônia durante um período de duas semanas. Um grupo de imagem foi gerado manualmente, enquanto outro grupo foi produzido pela automatização PDI. Cada grupo de imagem foi entregue à equipe sem que os membros soubessem qual método havia sido utilizado para gerar as imagens, garantindo a imparcialidade do teste. Apenas uma pessoa da equipe ficou responsável pelo controle dessa distribuição.

Ao final de cada dia de análise, os usuários forneciam uma avaliação da qualidade da imagem utilizada, atribuindo uma nota e fazendo observações sobre as imagens com as quais trabalharam.

Após a conclusão do teste, foi constatado que as imagens geradas pela automatização apresentavam uma qualidade equiparável às imagens geradas manualmente, sendo, em muitos casos, até superiores, com mais detalhes visíveis.

## **4 RESULTADOS**

### **4.1- Desempenho e Eficiência:**

Com a automatização do processo de PDI, houve um ganho significativo no tempo de aquisição das imagens no site e a disponibilização das imagens prontas para a equipe. anteriormente, para processar uma única imagem, representando uma órbita-ponto, o usuário precisava acessar o site, baixar as imagens banda a banda, realizar a composição colorida, gerar as frações de solo, sombra e vegetação, aplicar o contraste em cada uma delas, renomear e finalmente disponibilizar essas imagens. Esse processo manual levava cerca de 15 minutos por imagem.

Considerando que são processadas três imagens por dia, o tempo total estimado para o processamento manual era de, no mínimo, 45 minutos diários. Com a implementação da automatização do PDI, todas essas etapas são realizadas, para as mesmas três imagens, em apenas 15 minutos, maximizando a eficiência.

Além da redução no tempo de processamento, as imagens geradas pela automatização mantiveram uma qualidade excelente, tornando o processo não apenas mais rápido, mas também confiável para o uso diário pela equipe.

### **4.2- Qualidade dos Resultados:**

A automatização do processo de PDI não só otimizou o tempo de processamento das imagens, como também garantiu a consistência e a qualidade dos resultados. As imagens geradas automaticamente passaram por um rigoroso processo de validação, no qual foram comparadas com aquelas produzidas manualmente. Nesse processo, verificou-se que a qualidade das frações geradas (solo, sombra e vegetação) foi mantida em alto nível, sendo que, em alguns casos, as imagens automatizadas apresentaram até mais detalhes e precisão.

Além disso, o processo automatizado eliminou a possibilidade de erros humanos, como falha na aplicação de contraste ou na composição de bandas, que podem ocorrer em um fluxo manual. Isso assegura que as imagens entregues à equipe estejam sempre

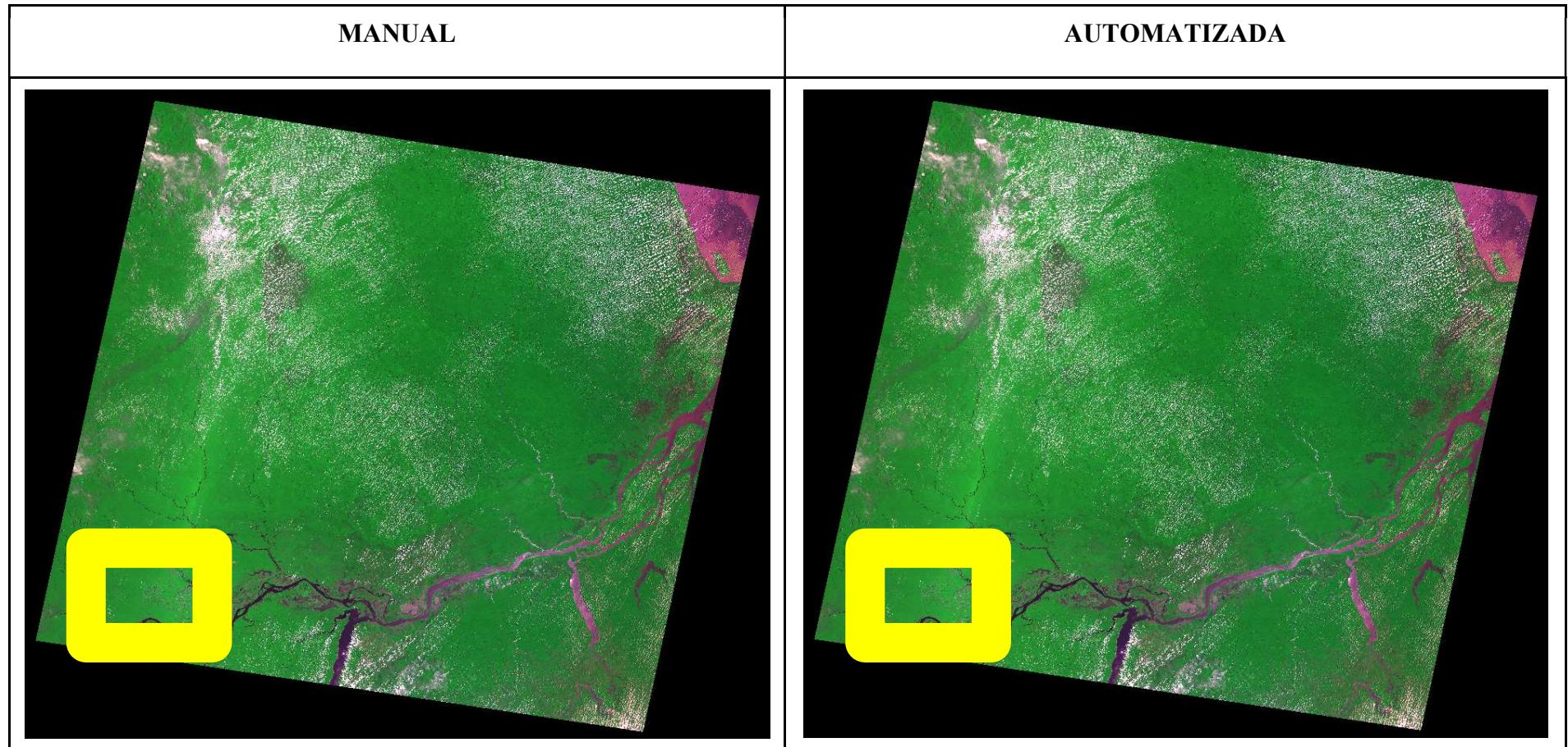
dentro do padrão de qualidade exigido para as análises. A consistência dos resultados reforça a confiança da equipe no uso dessas imagens para tomadas de decisão, permitindo que o processo seja repetível e confiável em todas as execuções.

#### **4.3- Exemplos de Saídas:**

A seguir, são apresentados exemplos de imagens processadas pela automatização do PDI, comparando-as com as imagens geradas manualmente por intérpretes. Cada conjunto de imagens inclui a composição RGB, bem como as frações resultantes de solo, sombra e vegetação.

Esses exemplos demonstram que a automatização não apenas iguala, mas em muitos casos, supera o processo manual, garantindo maior detalhamento e precisão na geração das frações e análise das imagens. A comparação visa ilustrar a eficiência da automatização, destacando como ela mantém os padrões de qualidade e oferece imagens confiáveis.

Tabela 1: Composição RGB manual X automatizada (órbita-ponto: 037\_015).



A área quadrada destacada em amarelo será utilizada para realizar as comparações entre as frações geradas manualmente e automaticamente, utilizaremos essa área com zoom para melhor detalhamento de visualização.



Tabela 2: Fração solo manual X automatizada (órbita-ponto: 037\_015).

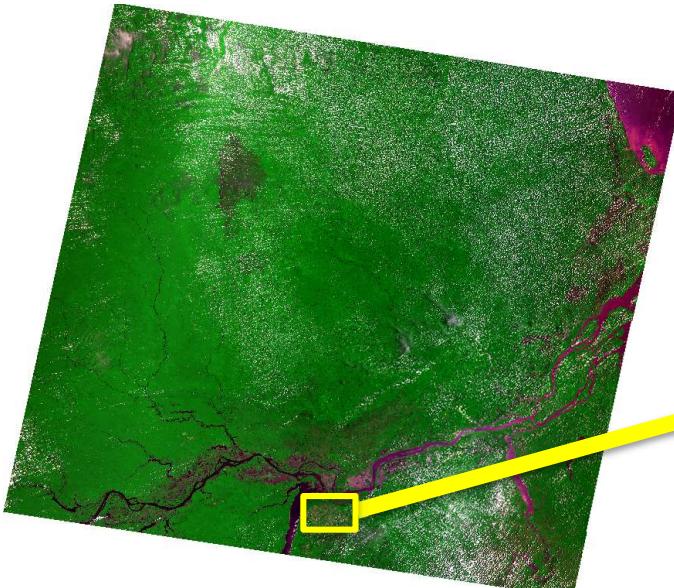
RGB normal	Área com zoom
	

Tabela 3: Fração vegetação manual X automatizada (órbita-ponto: 037\_015).

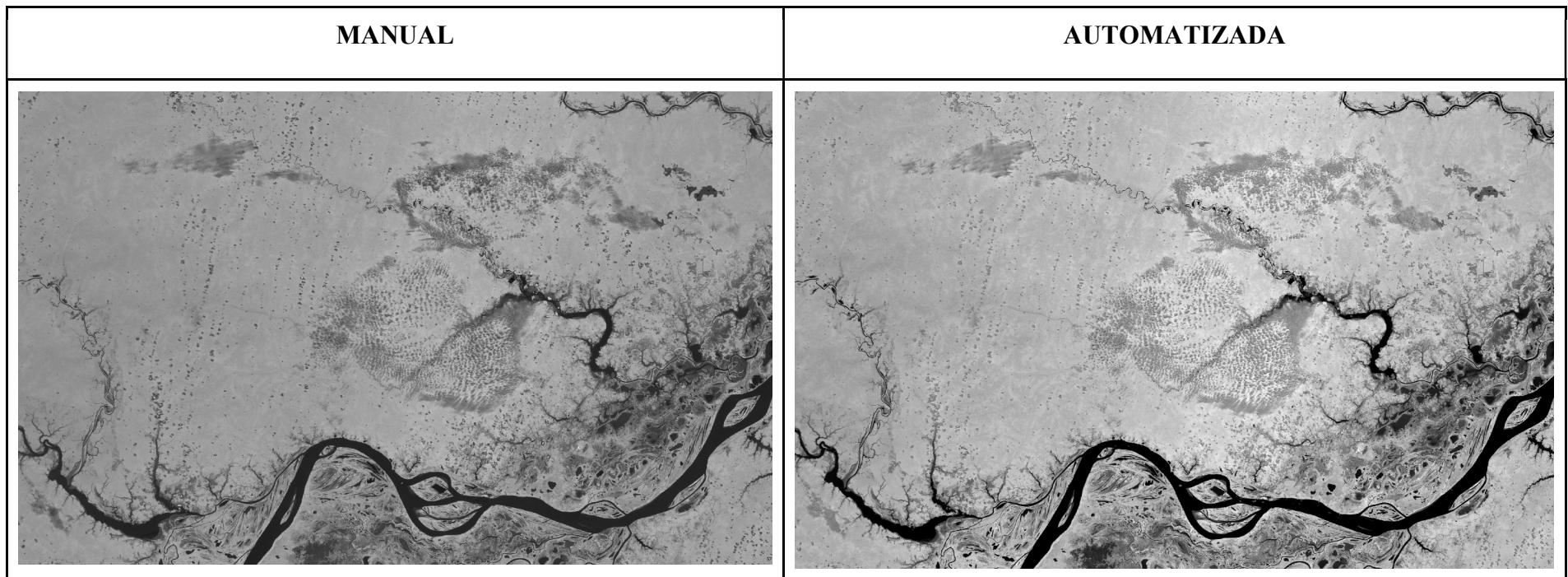


Tabela 4: Fração sombra manual X automatizada (órbita-ponto: 037\_015).

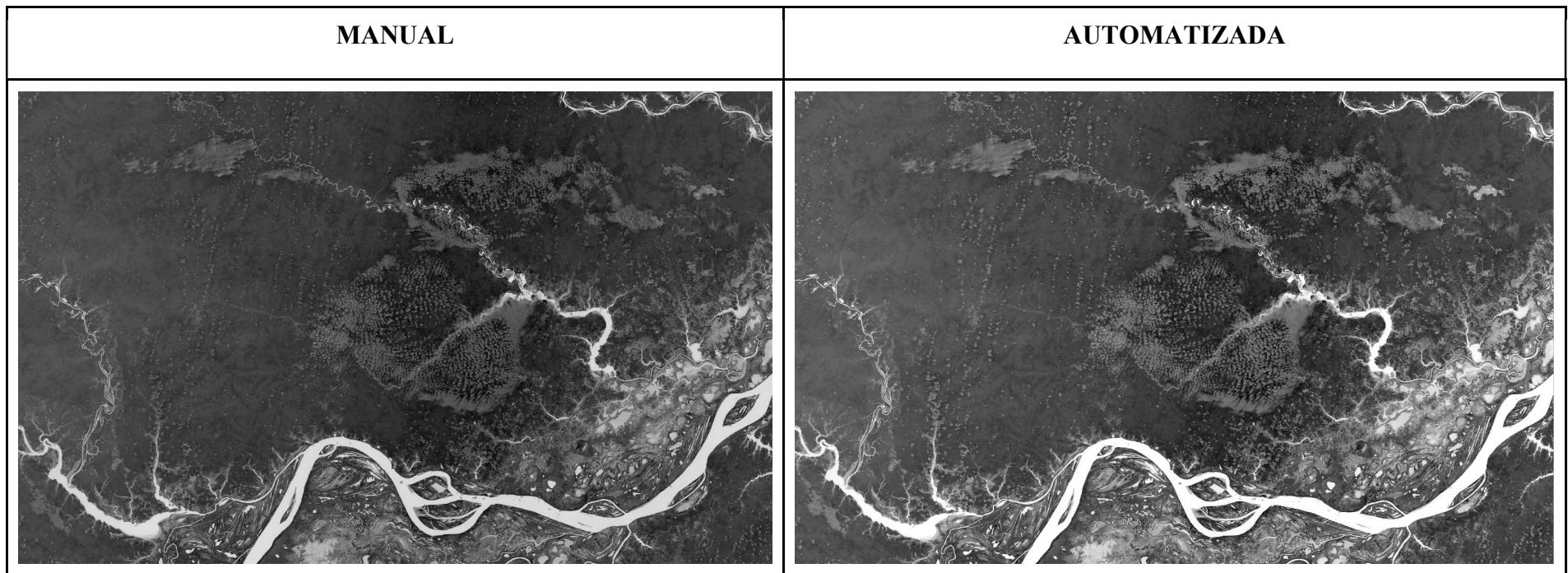


Tabela 5: Fração solo com contraste manual X automatizada (órbita-ponto: 037\_015).

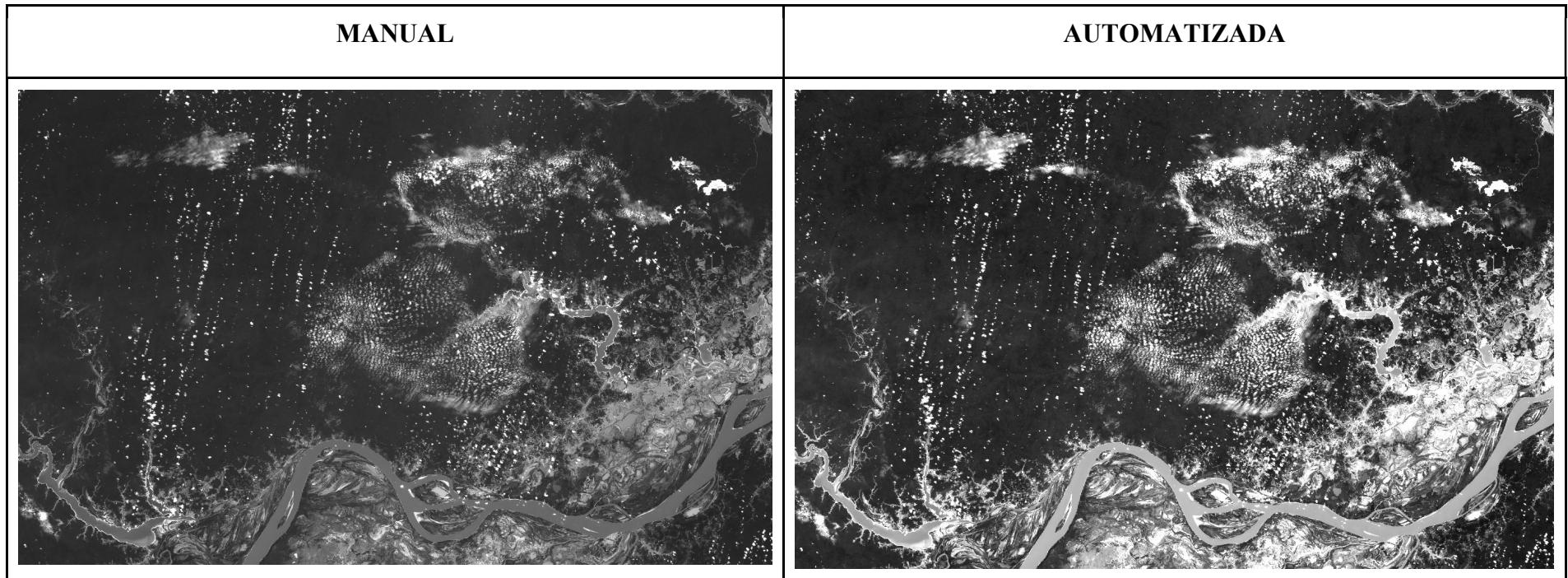


Tabela 6: Fração vegetação com contraste manual X automatizada (órbita-ponto: 037\_015).

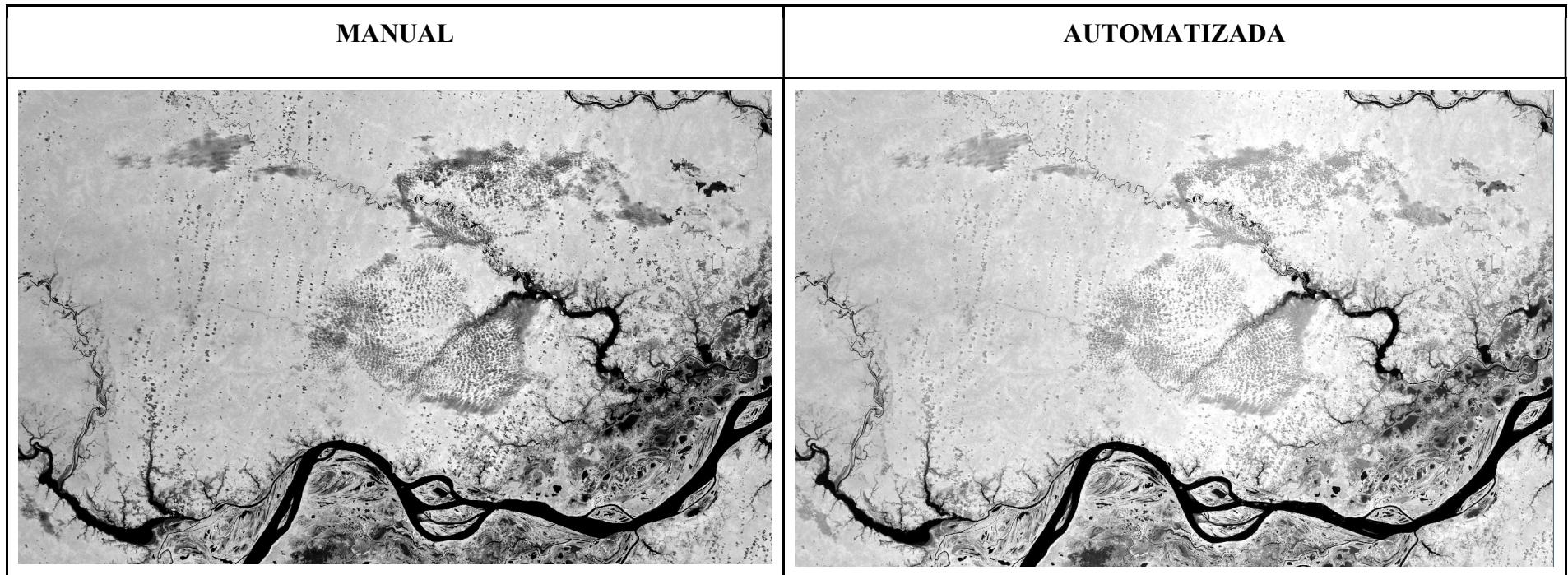
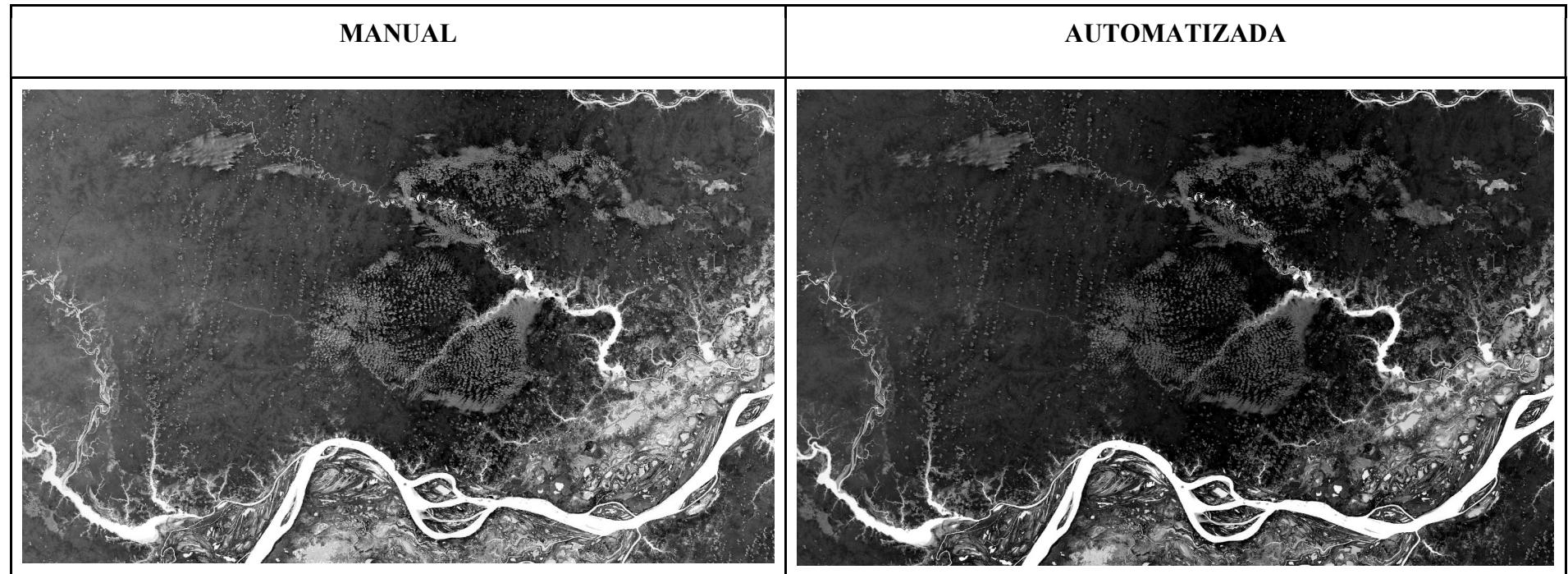


Tabela 7: Fração sombra com contraste manual X automatizada (órbita-ponto: 037\_015).



## **5 DESAFIOS E LIMITAÇÕES**

Durante o desenvolvimento e implementação do processo de PDI com o satélite Amazônia-1, foram encontrados diversos desafios e limitações que impactaram o fluxo de trabalho e os resultados esperados.

Um dos principais obstáculos é que o satélite Amazônia-1 não possui algoritmo dedicado para realizar uma correção atmosférica, o que impede a padronização eficaz do PDI em cenas coloridas. A ausência dessa correção dificulta a uniformidade entre diferentes capturas, criando inconsistências que precisam ser corrigidas manualmente.

Outro desafio significativo enfrentado, foi que inicialmente optamos por utilizar uma abordagem que envolvia a coleta de vários pontos espectrais de uma mesma imagem, e repetimos essa mesma técnica para todas as imagens que representam o bioma Amazônia, com o objetivo de calcular uma média global para as frações (solo, sombra e vegetação). No entanto, essa estratégia mostrou-se inviável, pois a variabilidade espectral entre diferentes órbitas-ponto era muito grande, tornando difícil obter resultados precisos. Com isso, aprimoramos a abordagem para obter dados espectrais específicos para cada órbita-ponto. Dessa forma, o MLME passou a utilizar informações específicas de cada órbita, resultando em uma maior precisão no cálculo das frações.

Na etapa de aplicação do contraste nas frações geradas, encontramos outro desafio, pois começamos utilizando a técnica de min/max, que se mostrou inadequada para todas as frações, onde sempre, uma das frações apresentavam falhas, gerando inconsistência nos resultados. Para solucionar esse problema, implementamos um contraste separado para cada uma das frações, ajustando por porcentagem, o que melhorou significativamente a qualidade visual e a precisão das imagens.

Esses desafios e limitações evidenciam a complexidade de automatizar processos de PDI e a necessidade de constantes ajustes e refinamentos para superar as barreiras tecnológicas e garantir resultados de alta qualidade.

## **6 CONCLUSÃO**

A automatização do processamento digital de imagens (PDI) e a aplicação do Modelo Linear de Mistura Espectral (MLME) trouxeram inúmeros benefícios para o fluxo de

trabalho do DETER Amazônia. Entre os principais ganhos, destacam-se a significativa redução no tempo de processamento e a melhoria na eficiência das operações diárias, permitindo que a equipe tenha acesso a imagens prontas para análise em um período muito menor. Esse processo, que antes demandava um esforço manual de cerca de 45 minutos por conjunto de imagens, agora é realizado automaticamente em aproximadamente 15 minutos, gerando imagens com qualidade e confiabilidade.

A aplicação do MLME, aliada à automatização, também garantiu uma precisão considerável na separação das frações de solo, sombra e vegetação, otimizando a análise e interpretação das áreas monitoradas. Além disso, a confiabilidade dos resultados gerados pela automatização tem permitido à equipe trabalhar com maior agilidade e segurança.

Quanto às melhorias futuras, uma das principais sugestões é o desenvolvimento de ferramentas que integrem as informações do sistema PRODES com outros tipos de informações, como máscaras de nuvens e sombras, para aprimorar a precisão na identificação de áreas de desmatamento. Com essas informações e ferramentas alinhadas, é possível gerar alertas automatizados, o que aumentaria ainda mais a eficiência no monitoramento da região amazônica. A evolução dessas funcionalidades permitirá maior agilidade no combate ao desmatamento, além de facilitar a tomada de decisões estratégicas de forma mais rápida e precisa.

## **7 AGRADECIMENTOS**

Os autores agradecem o apoio do Programa de Capacitação Institucional, uma parceria INPE e CNPq, através do Processo Nº 400077/2022-1 e Processo individual Nº 384895/2023-9, 380895/2024-2 e 99/2023/SEI-INPE

## **8 REFERÊNCIAS**

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **AMAZONIA 1: Descritivo da Missão e do Satélite.** São José dos Campos: INPE, 2021. Disponível em: <<https://www.gov.br/inpe>>. Acesso em: 13 Sep. 2024.

**A explicação, o motivo e a razão para os testes A/B e multivariado.** Oracle Brasil. Disponível em: <<https://www.oracle.com/br/cx/marketing/what-is-ab-testing/>>. Acesso em: 15 Sep. 2024.

BOSCOLO, Helena K.; KÖRTING, T.; NAMIKAWA, L.; *et al.* Contributions of the Brazil's National Institute for Space Research (INPE) to emergency response in the International Space and Major Disasters Charter. **Discover Applied Sciences**, 2024.

LEMENKOVA, P.; DEBEIR, O. Satellite Image Processing by Python and R Using Landsat 9 OLI/TIRS and SRTM DEM Data on Côte d'Ivoire, West Africa. **Journal of Imaging**, v. 8, n. 12, p. 317, 24 nov. 2022.