# PIBIC-PIBITI/CNPq/INPE
# RELATÓRIO TÉCNICO DE ATIVIDADES

<v9>

**Número do Processo Institucional:**138924/2021-0

**Número do Processo Individual:137289/2021-9**

**Bolsista:** Jonatas dos Reis Ferreira

**Orientador:** Claudio Clemente Faria Barbosa

**Coorientador: (quando houver)**

**Área:**
Desenvolvimento de rotinas para o gerenciamento da base de dados radiométricos na plataforma MAPAQUALI de monitoramento de sistemas aquáticos por Sensoriamento Remoto.

**Vigência original da bolsa:** 01/09/2021 a 31/08/2022

**Modalidade da bolsa:** PIBIC

# RELATÓRIO TÉCNICO

## 1) Resumo do Projeto

Trabalho relacionado a bolsa de iniciação cientifica junto ao Laboratório de Instrumentação para Sistemas Aquáticos (http://www.dpi.inpe.br/labisa/ - labISA), criado no ano de 2013, visando trabalhar com a implementação de algoritmos e conversão de rotinas (R para Python) para análise de imagens de satélite de ambientes aquaticos no sistema MAPAQUALI, com o objetivo de: 1) Desenvolver algoritmos para gerenciamento de corpos d'água, 2) Utilizar algoritmos já existentes e converte-los para outra linguagem de forma que seja aprimorada a eficiência e sua utilidade, 3) Estudo da estrutura e arquitetura do sistema MAPAQUALI ( GeoServer, rotinas Python, rotinas R, algoritmos matemáticos, algoritmos de Machine Learning (Random Forest) além de estudar os tipos de imagem de satélite, bandas, localização e instrumentação.

## 2) Objetivo

Implementar e converter rotinas em linguagem R para Python, no contexo do projeto MAPAQUALI de monitoramento de qualidade da água por imagensde satélite. Desenvolver algoritmos em ambiente Python para estimativa de indicadores qualidade da água usando modelos de TSS e TSI e algoritmo de Random Forest de aprendizado de máquina, e também estudo e aprimoramento pessoal na área de sensoriamento remoto e desenvolvimento da lógica de programação e habilidade de solução de problemas com a finalidade de aprimorar a funcionalidade e eficiência das rotinas.

## 3) Atividades Desenvolvidas durante o período da bolsa

No início da bolsa de iniciação foram propostas as seguintes atividades: 1) Estudo das rotinas e ínicio da conversão R para Python 2) Estudo das bibliotecas e funções necessárias para a implementação das rotinas em Python. 3) Implementação das rotinas para integracao no pipeline do projeto MAPAQUALI. Primeiramente foram implementadas as rotinas de TSS para cálculo de quantidade do Total de solidos suspensos, e TSIdo Total de solidos inorganicos suspensos . Também foram desenvolvidas as rotinas de determinação de profundidade do sensor secchi, coeficiente de atenuação difusa (KD), algoritmo de predição de concentração de ficocianina e clorofila.

*Figura 1 – Imagem LANDSAT original*

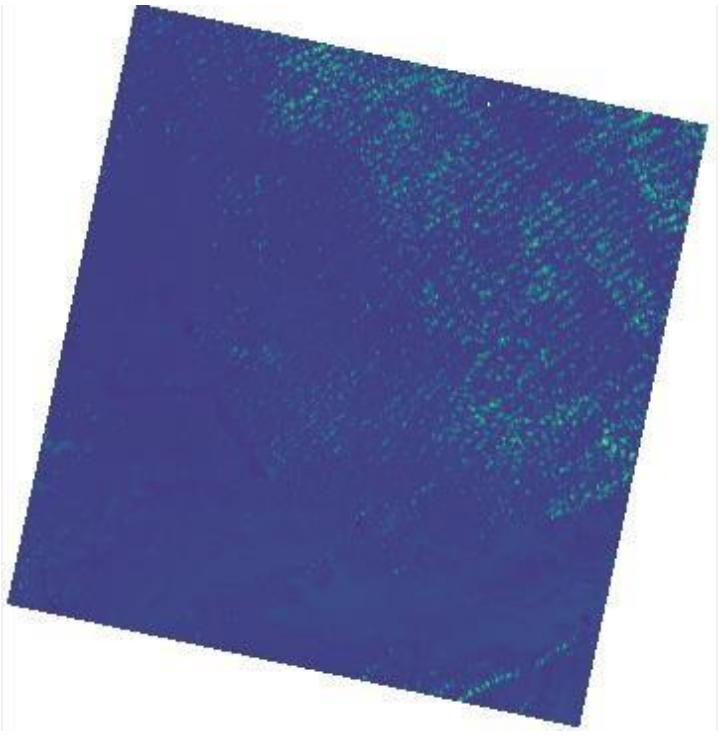

Figura 2 – Corpo d'agua recortado com base num shapefile já definido através de um algoritmo Python. (Lago Curuai)

```python
def recortar(diretorio_imagem: str, shp: str, diretorio_destino: str):
    """Recorta uma número n de imagens com base num shapefile

    Args:
        diretorio_imagem (str): diretorio da imagem a ser recortada
        shp (str): shapefile correspondente
        diretorio_destino (str): path de destino da imagem recortada
    """

    bandas = ImagemProcessBatch.procurar_bandas(diretorio_imagem)
    for banda in bandas:

        nome_banda = Path(banda).stem
        recorte = ImagemProcess.recortar(banda, shp)
        ImagemProcess.salvar_imagem(recorte, diretorio_destino, nome_banda + '_recortada' + '.TIF')
```

*Figura 3 - Algoritmo TSS e seu produto.*

```python
def tss(img: str, img_dst: str) -> str:
    """Função que aplica algoritmo TSS na imagem e salva"""
    np.seterr(divide='ignore')

    try:
        with rasterio.open(img) as ds:
            banda_lida = ds.read()
            tss_array = np.exp(9.656+1.672*np.log(abs(banda_lida)))
            out_meta = ds.meta
            out_meta.update({"driver": "GTiff", "dtype": 'float32'})

            out_path = Path(img_dst, Path(img).stem + '_TSS.TIFF')


        with rasterio.open(out_path, "w", **out_meta) as dest:
                dest.write(tss_array)
        return out_path

    except EnvironmentError:
        raise OSError('Arquivo corrompido ou tipo de arquivo inválido')
```

*Figura 4 - Algoritmo TSI e seu produto.*

```python
def tsi(img: str, img_dst: str) -> str:
    """Função que aplica algoritmo TSS na imagem e salva"""
    np.seterr(divide='ignore')

    try:
        with rasterio.open(img) as ds:
            banda_lida = ds.read()
            tsi_array = np.exp(10.73+2.08 * np.log(abs(banda_lida)))
            out_meta = ds.meta
            out_meta.update({"driver": "GTiff", "dtype": 'float32'})

            out_path = Path(img_dst, Path(img).stem + '_TSI.TIFF')

        with rasterio.open(out_path, "w", **out_meta) as dest:
                dest.write(tsi_array)
        return out_path
    except EnvironmentError:
        raise OSError('Arquivo corrompido ou tipo de arquivo inválido')
```

*Figura 5 – Algoritmo de profundidade Secchi e seu produto.*

```
class RF_secchi:
    def __init__(self):
        self.train_df = []
        self.rf_x = []
        self.rf_y = []
        self.sentinel2Bands = []
        self.img_metadata = {'rows':[],
                             'cols':[],
                             'bands':[],
                             'geo_transform':[],
                             'projection':[]
                             }
        self.x_data_nan = []

    def preload_imgs_data(self,path_list):
        self.sentinel2Bands = [gdal.Open(b, gdal.GA_ReadOnly) for b in path_list]
        self.img_metadata['rows'] = self.sentinel2Bands[0].RasterYSize
        self.img_metadata['cols'] = self.sentinel2Bands[0].RasterXSize
        self.img_metadata['bands'] = self.sentinel2Bands[0].RasterCount
        self.img_metadata['geo_transform'] = self.sentinel2Bands[0].GetGeoTransform()
        self.img_metadata['projection'] = self.sentinel2Bands[0].GetProjectionRef()

    def prepare_imgs_toRF(self):
        bands_dict = {}
        # Banda 2
        bands = ['B2','B3','B4','B5','B6']
        for band,i in zip(self.sentinel2Bands,bands):
            bands_dict[i] = (band.ReadAsArray() / np.pi) / 10000
def Load_train_data(self,path,delimiter=','):
    if path.endswith(".csv"):
        self.train_df = pd.read_csv(path, delimiter=delimiter)
    if path.endswith(".xlmx"):
        self.train_df = pd.read_excel(path, delimiter=delimiter)
    self.rf_x = self.train_df.loc[:, ['B2', 'B3', 'B4', 'B5', 'B6', 'b3_b2', 'b3_b4', 'b3_b6', 'ndci']].values
    self.rf_y = self.train_df['secchi']

def save_rf_trees(self,path='./my_random_forest.joblib'):
    joblib.dump(self.rf_trees,path)

def load_rf_trees(self,path='./my_random_forest.joblib'):
    self.rf_trees = joblib.load(path)

def apply_RF_training(self,save=True,path='./my_random_forest.joblib',n_estimators=62, max_features = 6, random_state=0):
    self.rf_trees = RandomForestRegressor(n_estimators=n_estimators, max_features = max_features, random_state=random_state).fit
    (self.rf_x,self.rf_y)
    if save == True:
        self.save_rf_trees(path)

def apply_RF_prediction(self):
    prediction = self.rf_trees.predict(self.x_data_nan)
    self.img_prediction = prediction.reshape((self.img_metadata['rows'], self.img_metadata['cols']))
```

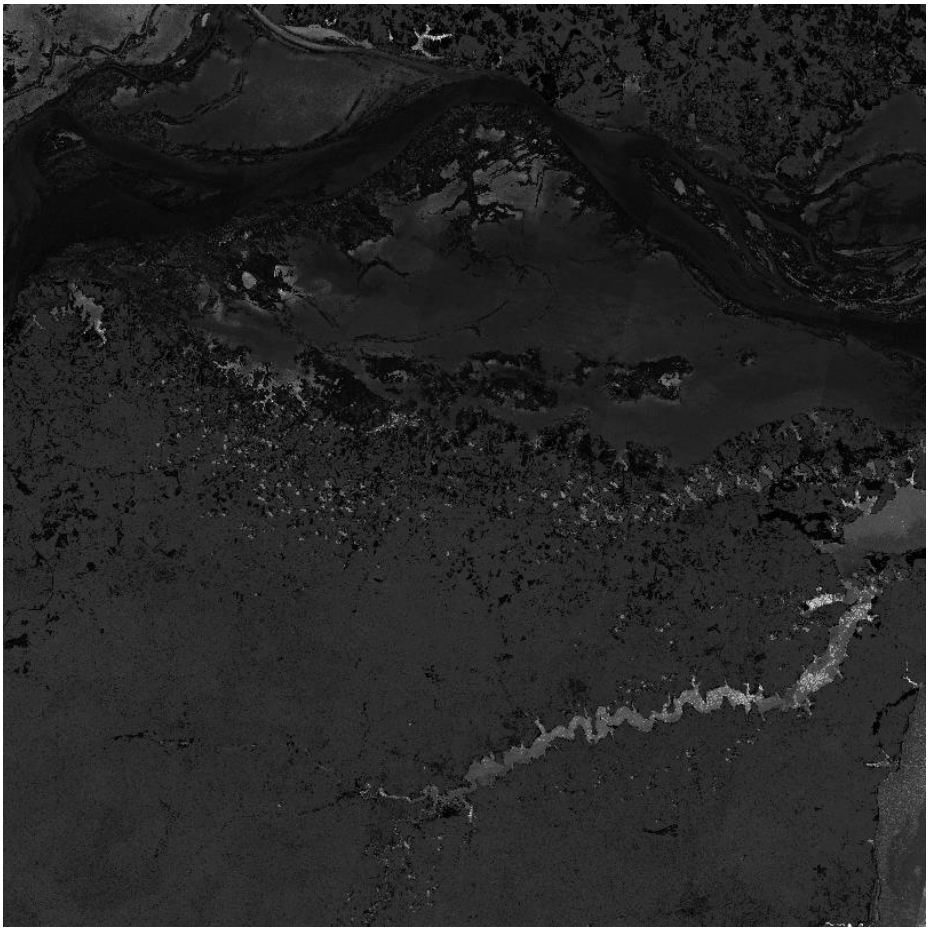

*Figura 6 – Algoritmo de KD e seu produto para um lago da planicie Amazônica .*

```
class KD_mapaquali:
    """Classe para realizar cálculos que compõem o KD
    """

    def __init__(self):
        """Inicia a classe criando as características das imagens
        """
        self.water_abs_bb = pd.read_csv(r"rotinas\kd\w_and_bb_s2a.csv")

        # Elements for equation
        self.w = self.water_abs_bb.iloc[2,3]
        self.j = 0.0604
        self.k = 0.0406
        self.l = 0.0402
        self.m = 0.1310

        self.sentinel2Bands = []
        self.img_metadata = {'rows':[],
                             'cols':[],
                             'bands':[],
                             'geo_transform':[],
                             'projection':[]
                             }
```

```
#Kd 670

bb = self.bbp_670 + self.water_abs_bb.iloc[2,3]

ratio_bbw_bb = self.water_abs_bb.iloc[2,3]/bb

absorption = self.abs_660

# Semi-analitic model (KD)

kd_660 = (1+m0*teta_s)*absorption + (1-chi*ratio_bbw_bb) * m1 * (1 - m2 * np.exp(-m3*absorption))*(bb)

self.products_kd = {'490': kd_490, '560': kd_560, '670': kd_660}
```

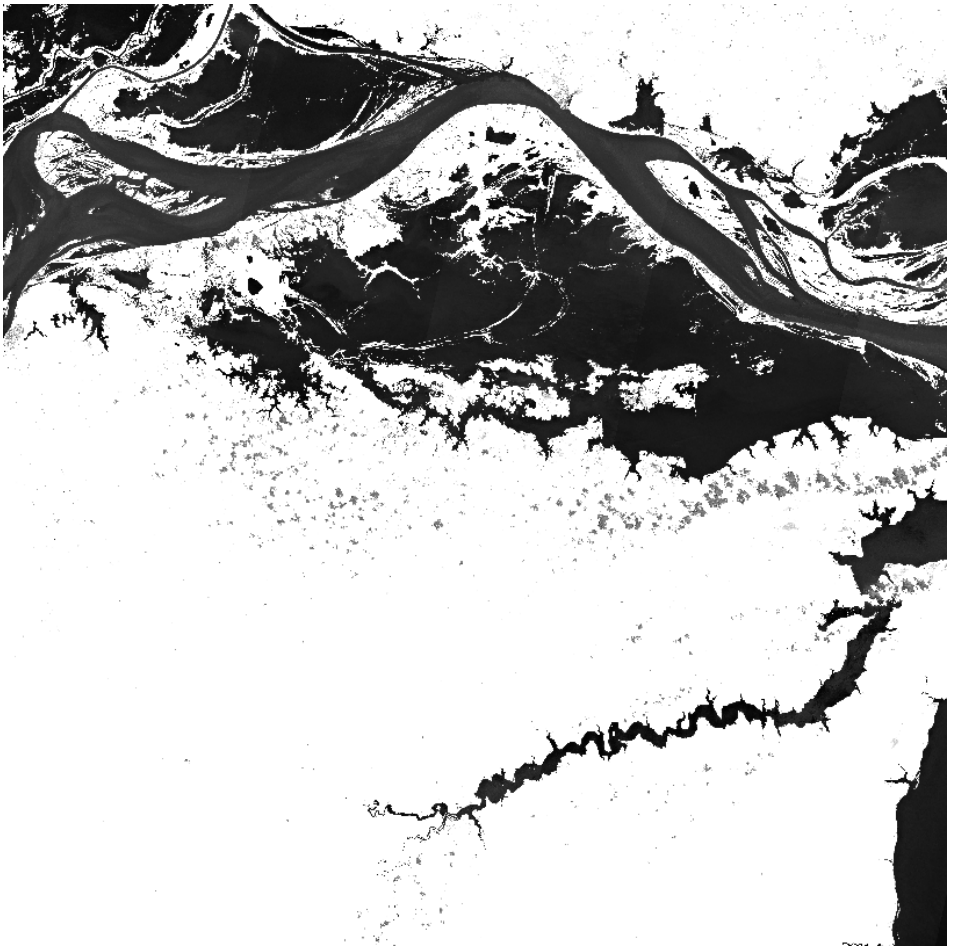

*Figura 7 – Algoritmo de Ficocianina e seu produto para a represa Billings.*

```
array_rf = np.stack((LH_1_613_array,ndpci_array,ndpci_2_array))

data_x = self.data_xgboost

data_y = self.data_pc.reshape(-1)

model_xgb = xgb.XGBRegressor(objective='reg:squarederror',learning_rate=0.05, max_depth= 2, n_estimators=80,
random_state=12).fit(data_x,data_y)

x = np.moveaxis(array_rf, 0, -1)

x_data = x.reshape(-1,3)

x_data_nan = np.nan_to_num(x_data)

x_data_trans = self.scalor_x.transform(x_data_nan)

prediction = model_xgb.predict(x_data_trans)

real_prediction = (self.scalor_y.inverse_transform(prediction.reshape(-1,1)))

real_prediction[self.bands_dict['green'].flatten()==9999] = np.nan

self.img_prediction = real_prediction.reshape((self.img_metadata['rows'],self.img_metadata['cols']))

self.exp_img_prediction = np.exp(self.img_prediction)
```

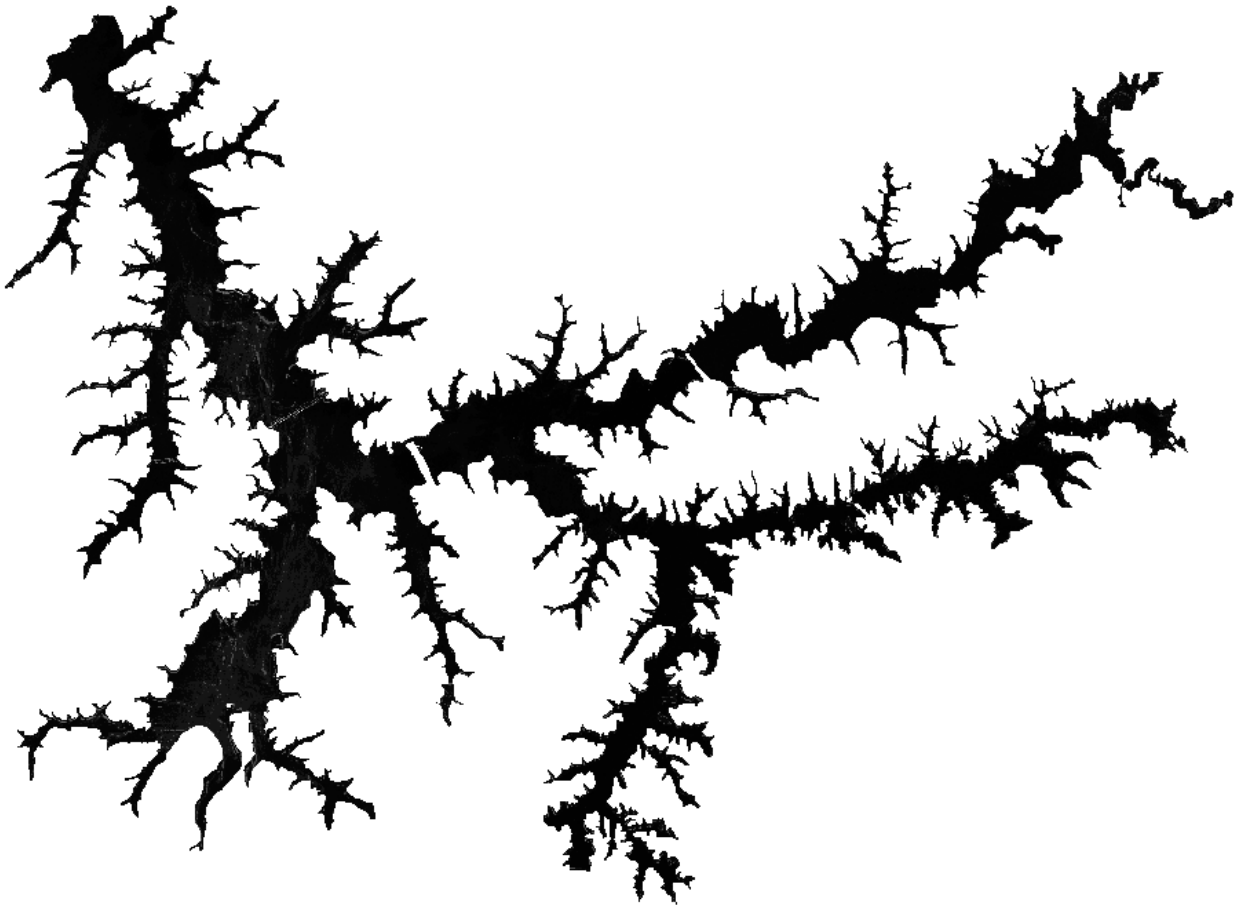

*Figura 8 – Algoritmo de Clorofila e seu produto.*

```
def calc(self):
    """Realiza alguns calculos entre as bandas e faz a predição do modelo com o xgboost
    """
    #Calcula LH
    LH_2_705_array = self.bands_dict['b5_array'] - (self.bands_dict['b7_array'] + ((self.bands_dict['b3_array'] - self.
    bands_dict['b7_array']) * ((783-705)/(783-560))))

    #Calcula NDPCI
    ndci_array = (self.bands_dict['b5_array'] - self.bands_dict['b4_array']) / (self.bands_dict['b5_array'] + self.bands_dict
    ['b4_array'])

    LH_2_705_array[LH_2_705_array==0] = 9999

    ndci_array[ndci_array==0] = 9999


    #stack array
    array_rf = np.stack((ndci_array,LH_2_705_array))

    data_x = self.data_rf

    data_y = self.data_chla.reshape(-1)

    model_rf = RandomForestRegressor( max_depth=4, n_estimators=200 , random_state=12).fit(data_x,data_y)
    # model_rf = RandomForestRegressor( max_depth=4, n_estimators=200 , random_state=12).fit(data_x.values,data_y.values)

    x = np.moveaxis(array_rf, 0, -1)

    x_data = x.reshape(-1,2)

    x_data_trans = self.scalor_x.transform(x_data)

    prediction = model_rf.predict(x_data_trans)

    #ATENÇÃO PARA SAIR EM VALOR DE PC COLOCAR FUNÇÃO NP.EXP
    real_prediction = np.exp(self.scalor_y.inverse_transform(prediction.reshape(-1,1)))
```

## 4) Resultados Obtidos em função do Plano de Trabalho proposto

1-Processamento de diferentes tipos de dados de sensoriamento remoto; 2-Execução e aplicação de diferentes lógicas de rotina num ambiente de desenvolvimento para pesquisa; 3- Com base na forma de trabalho proposta no ambiente MAPAQUALI, onde foram definidas diversas tarefas, o aprendizado alcançado foi satisfatório tanto para o aluno quanto para o laboratório, onde todos os objetivos propostos e os estudos direcionados foram concluídos no periodo de vigência da bolsa de iniciação ciêntifica, além de ser uma adição interessante para o desenvolvimento pessoal quando se trata de evolução própria.

## 5) Conclusões Gerais

Foi possível ter um conhecimento basico sobre a área de sensoriamento remoto, principalmente quando se trata de análise de dados de campos, de corpos d'água, além de aprimorar a parte técnica de programação, geoprocessamento, e adquirir conhecimento em algoritmos computacionais para fins de geração de produtos de pesquisa para entender e contextualizar como é a evolução dos corpos d'água ao longo do tempo no Brasil através do processamento de imagens de satélite.

19, **de** setembro **de 2022**

Jonatas dos Reis Ferreira

**Bolsista: Jonatas dos Reis Ferreira**

Claudio Clemente Faria Barbosa

**Orientador(a): Claudio Clemente Faria Barbosa**