Variedade de *Landforms*

Importando pacotes e inicializando *geemap*

```
import os
import ee
import geemap
import matplotlib as mpl
import matplotlib.pylab as plt
from matplotlib.colors import ListedColormap, BoundaryNorm
```

```
geemap.ee_initialize()
```

Variedade de *landforms*

Landforms

Nós exploramos três metodologias de classificação de *landforms* (Anderson *et al.* 2016; Theobald *et al.* 2015; Iwahashi & Yamazaki 2022) para definirmos quais delas seriam usadas no cálculo da variedade de *landforms*. Primeiro, nós classificamos as *landforms* como Anderson *et al.* (2016), exceto a *landform* ***Flat at the bottom of steep slope***. Assim, nossas *landforms* foram:

- **3 - Cool Steep Slope**
- **4 - Warms Steep Slope**
- **5 - Cliff**
- **11 - Summit/Ridgetop**
- **13 - Slope Crest**
- **21 - Flat Hilltop**
- **22 - Gentle Slope Hilltop**
- **23 - Cool Sideslope**
- **24 - Warm Sideslope**

- **30 - Dry Flats**
- **32 - Valley/Toeslope**
- **39 - Moist Flats**
- **43 - Cool Footslope**
- **44 - Warm Sideslope**

Variáveis classificadoras das *landforms*

Nós classificamos as *landforms* pela **inclinação do relevo** (*slope*), **aspecto** (*aspect*), **Índice de Posição Topográfica** (*TPI*) e **Índice de Umidade** (*moisture index*). As variáveis foram discretizadas em classes e combinadas para comporem os tipos de *landforms*. As *landforms* são classificadas principalmente pelo *slope* e TPI (Figure 1). O *aspect* classifica as faces *quentes* ou *frias* do relevo e o *moisture index* classifica as áreas planas em secas ou úmidas.

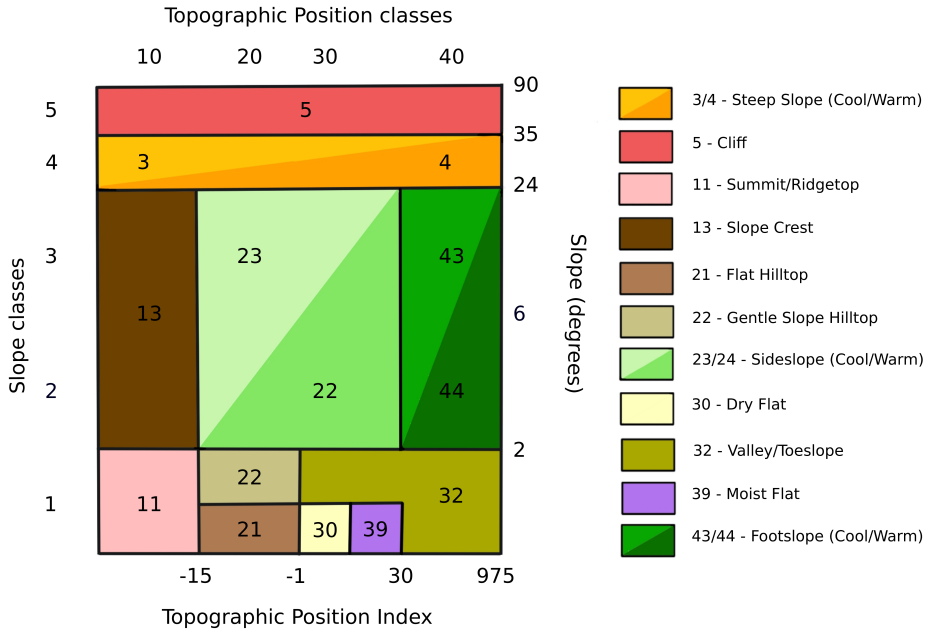

Figura 1. Classificação de landforms pela The Nature Conservancy para as paisagens na América

Cada variável foi discretizada em classes para comporem os tipos de *landforms*. Os limiares (Tabela 1) para a discretização foram definidos por ajustes visuais que melhor representavam as *landforms*.

Tabela 1. Descrição dos limiares de classificação de cada variável em classes.

Variáveis	Classes	Limiar inferior	Limiar superior
Inclinação do relevo	1	-1	2
Inclinação do relevo	2	2	6
Inclinação do relevo	3	6	24
Inclinação do relevo	4	24	35
Inclinação do relevo	5	35	90
TPI	1	-Inf	-15
TPI	2	-15	-1
TPI	3	-1	30
TPI	4	30	975
Aspecto	2	0	90
Aspecto	1	90	270
Aspecto	2	270	360
Índice de Umidade	0	-Inf	30000
Índice de Umidade	1	3000	Inf

Em seguida, as classes foram combinadas pela soma de cada classe multiplicada por um peso. O *moisture index* foi multiplicado por 1000, *aspect* por 100, TPI por 10 e *slope* por 1. Desta forma, o número resultante representa um código descrevendo as classes de cada variável. Por exemplo, 1231 é a classe 1 de *moisture index*, 2 de *aspect*, 3 de TPI e 1 de *slope*. Posteriormente, os valores finais foram convertidos em tipos de *landforms*, seguindo a Tabela 2.

Tabela 2. Critério de conversão dos códigos da combinação de classes em tipos de landforms.

Código	Tipos de *landforms*
10	11
11	11
12	11
13	13
14	11
15	5
20	21
21	21
22	22
23	24
24	24
25	5
31	30
32	32

Código	Tipos de *landforms*
33	24
34	24
35	5
40	32
41	32
42	32
43	43
44	3
45	5
51	51
111	11
112	11
113	13
114	3
115	5
121	21
122	22
123	23
124	3
125	5
131	30
132	32
133	23
134	3
135	5
141	32
142	32
143	43
144	3
145	5
151	51
211	11
212	11
213	13
214	4
215	5
221	21
222	22
223	24
224	4
225	5

Código	Tipos de *landforms*
231	30
232	32
233	24
234	4
235	5
241	32
242	32
243	44
244	4
245	5
251	51
1000	39

Bases de dados

Nós utilizamos para a classificação das *landforms* o Modelo Digital de Elevação (DEM) do Merit-DEM (Yamazaki *et al.* 2017), o acúmulo de fluxo do Merit-Hydro (Yamazaki *et al.* 2019) e a camada de uso do solo do MapBiomas (MapBiomas Project 2020). O Modelo Digital de Elevação possui uma resolução de 90 metros e foi escolhido por ser um produto global ao combinar dados dos satélites do *Shuttle Radar Topography Mission* (SRTM) (Farr *et al.* 2007) e *Advanced Land Observing Satellite* (ALOS) (Tadono *et al.* 2014), permitindo a replicabilidade da metodologia em outras regiões. O Merit-DEM corrige viéses de Modelo Digitais de Elevação gerados por imagens de satétite como *speckle noise*, *stripe noise*, *absolute bias* e *tree height bias* (Yamazaki *et al.* 2017). A correção de *tree height bias* é principalmente importante para a Floresta Amazônica devido à sua densidade de árvores altas. Além disso, há um produto derivado, o Merit-Hydro, que disponibiliza o acúmulo de fluxo global, que demandaria grande esforço computacional para ser calculado para todo o Brasil. O Merit-Hydro corrige os efeitos de densidade de árvores no cálculo da rede dendrítica, o que é importante para a Amazônia.

Nós incluímos as classes de água do MapBiomas para complementar a superfície gerada pelo acúmulo de fluxo na definição de áreas planas úmidas. O MapBiomas é um projeto nacional de mapeamento e classificação de mudanças do uso do solo dos últimos 30 anos a partir de dados de sensoriamento remoto.

Códigos para a criação da variedade de *landforms*

Nossas análises foram rodadas no *Google Earth Engine* (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote **geemap** (Wu 2020) em *Python* (Python Software Foundation 2023) como interface pela facilidade na documentação e reprodutividade das análises.

Inclinação do relevo (*slope*)

Nós criamos a superfície de *slope* a partir do Merit-DEM.

```
# Importando Modelo Digital de Elevação

DEM = ee.Image("MERIT/DEM/v1_0_3")

# Calculando o slope

slope = ee.Terrain.slope(DEM)
```

Aspecto (*aspect*)

Nós calculamos o *aspect* do relevo utilizando o mesmo DEM.

```
aspect = ee.Terrain.aspect(DEM)
```

Índice de Posição Topográfica (TPI)

Nós calculamos o Índice de Posição Topográfica (TPI)(Weiss 2001) para cada célula do *raster* dentro de um *kernel* circular com 7, 11 e 15 células de raio. O TPI é a diferença média de elevação entre a célula focal e um conjunto de células vizinhas.

$TPI = \frac{\sum_i^n (vizinhana_i - focal)}{n}$

a vizinhança i representa cada uma das n células dentro do *kernel* da célula focal. O índice final é composto pela média de TPI das três janelas, o que permite a consideração de diferentes níveis de resolução da paisagem, tanto local quanto regional (Theobald *et al.* 2015). Os tamanhos das janelas foram definidos visualmente para que melhor representassem as *landforms*, principalmente os *Summits*, *Valleys*, *Toeslopes* e *Hilltops* (*flat* e *gentle*). Os tamanhos das janelas também tinham que capturar os Platôs como *Summits*.

```
# Função para calcular o TPI

def calculate_TPI(pixel_size):

  # Calcule a média das células da vizinhança

  focal_mean = DEM.focalMean(**{
```

```
                        'radius': pixel_size,
                        'kernelType': "circle",
                        'units': "pixels"
                      })

  # Calcule a diferença entre a célula focal e média da região

  TPI = focal_mean.subtract(DEM)

  return TPI
```

```
# Tamanho das janelas

window_size = [7,11,15]

# Calculo do TPI para cada janela e calculo do TPI médio das janelas

TPI = ee.ImageCollection(list(map(calculate_TPI, window_size))).toBands().reduce("mean")
```

Índice de Umidade (*Moisture index*)

Nós calculamos o *moisture index* (Anderson *et al.* 2016) baseado no acúmulo de fluxo presente no Merit-Hydro (Yamazaki *et al.* 2019), na camada **upg**, que é calculado sobre o Merit-DEM. O *moisture index* é calculado da seguinte forma:

$moisture.index = \frac{\log(fluxo+1)}{(slope+1)} \times 1000$

onde fluxo é o acúmulo de fluxo e *slope* é o *slope* calculado anteriormente. O *moisture index* é a média do índice dentro de um *kernel* circular de uma célula de raio. O tamanho do raio foi escolhido visualmente para suavizar o índice, mas representando bem a distribuição dos cursos d'água.

```
# Importando o acúmulo de fluxo

flow_accumulation = ee.Image("MERIT/Hydro/v1_0_1").select("upg")

# Calculando o moisture index

moisture_index = (
    flow_accumulation
    .add(ee.Number(1))
```

```
    .divide(slope.add(ee.Number(1)))
    .log()
    .multiply(1000)
    .focalMean(**{'radius': 1,
                  'kernelType': "circle",
                  'units': "pixels"
                  }
                )
)
```

Convertendo variáveis em classes

Inclinação do relevo (*slope*)

As variáveis foram convertidas em classes utilizando a Tabela 1, seguindo Anderson *et al.* (2016). As classes de slope foram criadas com o seguinte código:

```
slope_classes = (
    slope
    .where(slope.gte(-1).And(slope.lte(2)), 1)
    .where(slope.gt(2).And(slope.lte(6)), 2)
    .where(slope.gt(6).And(slope.lte(24)), 3)
    .where(slope.gt(24).And(slope.lte(35)), 4)
    .where(slope.gt(35).And(slope.lte(90)), 5)
)
```

Aspecto (*aspect*)

O aspecto foi escolhido para definir as faces norte e sul do relevo no hemisfério sul.

```
aspect_classes = (
    aspect
    .where(aspect.gte(0).And(aspect.lte(90)), 2)
    .where(aspect.gt(90).And(aspect.lte(270)), 1)
    .where(aspect.gt(270).And(aspect.lte(360)), 2)
)
```

Índice de Posição do Relevo (TPI)

As classes de TPI foram definidas para representarem bem os *Summits*, *Valleys*, *Toeslopes* e *Hilltops*, que foram as *landforms* mais difíceis de ajustar os parâmetros.

```
TPI_classes = (
    TPI
    .where(TPI.lte(-15), 1)
    .where(TPI.gt(-15).And(TPI.lt(-1)), 2)
    .where(TPI.gte(-1).And(TPI.lte(30)), 3)
    .where(TPI.gt(30).And(TPI.lte(975)), 4)
)
```

Índice de Umidade (*moisture index*)

O limiar do índice umidade para classificar as áreas como umidas ou secas foram definidos visualmente para capturarem a distribuição dos cursos d'água sem criar áreas planas secas com excesso de ramificações dendríticas. Grandes rios (ex. Rio Amazonas, represas e lagos) não foram bem representados pelo *moisture index*, pois classificava somente a partes mais profundas como áreas úmidas, mantendo o restante dos grandes corpos d'água como regiões planas secas. Nós corrigimos essa classificação combinando a área úmidade classificada pelo acúmulo de fluxo com a camada de águas produzida pelo MapBiomas.

```
# Classificando o índice de umidade em classes

moisture_classes = (
    moisture_index.where(moisture_index.lte(3000), 0)
    .where(moisture_index.gt(3000), 1)
)

# Importando o dado de uso de solo do Mapbiomas e reprojetando para a escala do DEM

mapbiomas  = (
    ee.Image("projects/mapbiomas-workspace/public/collection7/mapbiomas_collection70_integ
    .select("classification_2020")
    .reproject('EPSG:4326', None, 92.76624)
)

# Reclassificando o raster do MapBiomas em água (1) e outras classes (0)

water = (
```

```
    mapbiomas
    .where(mapbiomas.eq(33), 1)
    .where(mapbiomas.neq(33), 0)
)

# Combinado o índice de umidade com a camada de água e reclassificando

moisture_classes = moisture_classes.add(water)

moisture_classes = (
    moisture_classes
    .where(moisture_classes.gte(1), 1)
    .where(moisture_classes.lt(1), 0)
)
```

Combinando as classes

Combinamos as classes para a geração de um código representativo de cada variável. O *moisture index* foi multiplicado por 1000, *aspect* por 100, TPI por 10 e *slope* por 1.

```
classes_collection = ee.Image([moisture_classes.multiply(ee.Number(1000)),
                               aspect_classes.multiply(ee.Number(100)),
                               TPI_classes.multiply(ee.Number(10)),
                               slope_classes])

landform_combination = classes_collection.reduce(ee.Reducer.sum())
```

Classificando os tipos de *landforms*

Classificamos os tipos de *landforms* pelo código gerado anteriormente e ajustamos visualmente alguns códigos para representarem bem as *landforms*. Por exemplo, o código 11 representa áreas de baixa inclinação do relevo e uma posição do relevo mais alta que o entorno, sendo portanto um topo de montanha (*Summit*). No entanto, alguns códigos tiveram que ser bem inspecionados para separar alguns tipos de *landforms* como *Sideslopes* de *Valleys* e *Toeslopes*.

```
landform_types = (
    landform_combination
    .mask(landform_combination.gt(0))
    .where(landform_combination.eq(10), 11)
```

```
    .where(landform_combination.eq(11), 11)
    .where(landform_combination.eq(12), 11)
    .where(landform_combination.eq(13), 13)
    .where(landform_combination.eq(14), 11)
    .where(landform_combination.eq(15), 5)
    .where(landform_combination.eq(20), 21)
    .where(landform_combination.eq(21), 21)
    .where(landform_combination.eq(22), 22)
    .where(landform_combination.eq(23), 24)
    .where(landform_combination.eq(24), 24)
    .where(landform_combination.eq(25), 5)
    .where(landform_combination.eq(31), 30)
    .where(landform_combination.eq(32), 32)
    .where(landform_combination.eq(33), 24)
    .where(landform_combination.eq(34), 24)
    .where(landform_combination.eq(35), 5)
    .where(landform_combination.eq(40), 32)
    .where(landform_combination.eq(41), 32)
    .where(landform_combination.eq(42), 32)
    .where(landform_combination.eq(43), 43)
    .where(landform_combination.eq(44), 3)
    .where(landform_combination.eq(45), 5)
    .where(landform_combination.eq(51), 51)
    .where(landform_combination.eq(111), 11)
    .where(landform_combination.eq(112), 11)
    .where(landform_combination.eq(113), 13)
    .where(landform_combination.eq(114), 3)
    .where(landform_combination.eq(115), 5)
    .where(landform_combination.eq(121), 21)
    .where(landform_combination.eq(122), 22)
    .where(landform_combination.eq(123), 23)
    .where(landform_combination.eq(124), 3)
    .where(landform_combination.eq(125), 5)
    .where(landform_combination.eq(131), 30)
    .where(landform_combination.eq(132), 32)
    .where(landform_combination.eq(133), 23)
    .where(landform_combination.eq(134), 3)
    .where(landform_combination.eq(135), 5)
    .where(landform_combination.eq(141), 32)
    .where(landform_combination.eq(142), 32)
    .where(landform_combination.eq(143), 43)
```

```
    .where(landform_combination.eq(144), 3)
    .where(landform_combination.eq(145), 5)
    .where(landform_combination.eq(151), 51)
    .where(landform_combination.eq(211), 11)
    .where(landform_combination.eq(212), 11)
    .where(landform_combination.eq(213), 13)
    .where(landform_combination.eq(214), 4)
    .where(landform_combination.eq(215), 5)
    .where(landform_combination.eq(221), 21)
    .where(landform_combination.eq(222), 22)
    .where(landform_combination.eq(223), 24)
    .where(landform_combination.eq(224), 4)
    .where(landform_combination.eq(225), 5)
    .where(landform_combination.eq(231), 30)
    .where(landform_combination.eq(232), 32)
    .where(landform_combination.eq(233), 24)
    .where(landform_combination.eq(234), 4)
    .where(landform_combination.eq(235), 5)
    .where(landform_combination.eq(241), 32)
    .where(landform_combination.eq(242), 32)
    .where(landform_combination.eq(243), 44)
    .where(landform_combination.eq(244), 4)
    .where(landform_combination.eq(245), 5)
    .where(landform_combination.eq(251), 51)
    .where(landform_combination.gte(1000), 39)
)
```

Exportando mapas para assets

```
# Nome do asset
assetId = "projects/ee-lucasljardim9/assets/landform_types"

# Importando mapa de biomas do IBGE para extrair as coordenadas mínimas e máximas do Brasi
bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")

def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()
```

```
# Extraindo a resolução do mapa
escala = landform_types.projection().nominalScale()
```

```
# Exportando para o gee
geemap.ee_export_image_to_asset(
    landform_types, description='landform_types', assetId=assetId, region=bioma_box, scale
)
```

Exemplo de *landforms*

Abaixo está uma representação das *landforms* na região de Alto Paraíso de Goiás-GO (Latitude:-14.11, Longitude:-47.26).

```
%%capture --no-display

# Delimitando a região
regiao = ee.Geometry.BBox(-47.4631, -13.9777, -47.1005, -14.1711)

# Creando a pasta para exportar as figuras
if not os.path.exists("figura"):

    os.mkdir("figura")

# Exportando a imagem da região
geemap.ee_export_image(
    landform_types, filename="figura/landform_types.tif", scale=escala, region=regiao, fil
)
```

```
# Paleta de cores das landforms
palette = [
    "#ffc408", # 3
    "#ffa101", # 4
    "#ef595a", # 5
    "#ffbdbe", # 11
    "#6e4100", # 13
    "#af7b53", # 21
    "#c8f6ad", # 23
    "#c8c284", # 22
    "#83e763", # 24
```

```
    "#08a702", # 43
    "#ffffbe", # 30
    "#a9a800", # 32
    "#b671f2", # 39
    "#0a7000" ] # 44

# Discretinzando a paleta de cores
cmap = ListedColormap(
   palette, 'Custom cmap')

class_bins = [3, 4, 5, 11, 13, 21, 22, 23, 24, 30, 32, 39, 43, 45]

norm = BoundaryNorm(class_bins,
                    13)

# Plotando mapa
geemap.plot_raster("figura/landform_types.tif", cmap = cmap, norm = norm, figsize = [20, 1
```

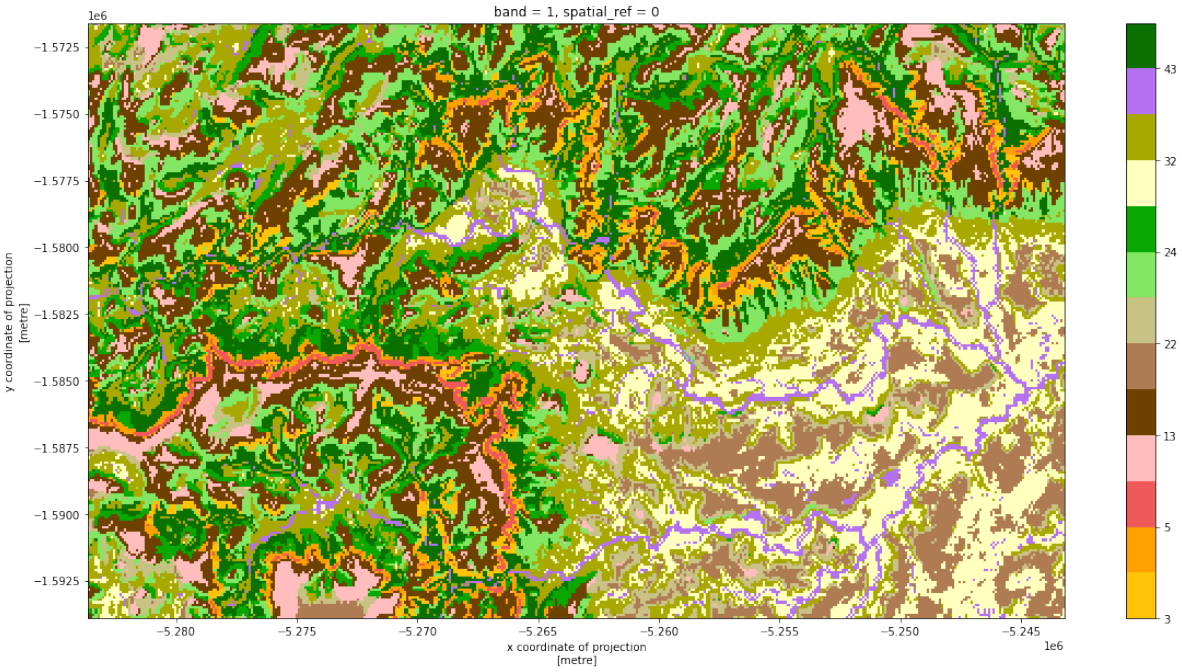

Figura 2. Landforms classificadas na região de Alto Paraíso de Goiás-GO, Brasil.

Calculando a variedade de *landforms*

A variedade de *landforms* foi calculada como a soma de tipos diferentes de *landforms* dentro de um *kernel* circular de uma célula focal. O tamanho do raio do *kernel* foi definido calculando a variedade em diferentes raios (2, 5, 7, 10, 15, 20 células) e calculando o ganho de variedade a cada aumento de raio. O raio escolhido foi aquele que o subsequente não adicionou variedade. Desta forma, o raio representa o nível de resolução da paisagem que captura o máximo de variedade de *landforms*. Raios maiores podem aumentar a variedade, mas devido a mudança de paisagem. Assim, o raio escolhido foi de 5 células de raio (450 metros) para todo o Brasil. Abaixo está uma representação da variedade de *landforms* para a mesma região de Alto Paraíso de Goiás-GO.

```
radius_pixels = 5

landform_variety = (
    landform_types
    .neighborhoodToBands(ee.Kernel.circle(radius_pixels))
    .reduce(ee.Reducer.countDistinct())
)
```

```
%%capture --no-display

geemap.ee_export_image(
    landform_variety, filename="figura/landform_variety.tif", scale=escala, region=regiao,
)
```

```
geemap.plot_raster("figura/landform_variety.tif", figsize = [20, 10])
```

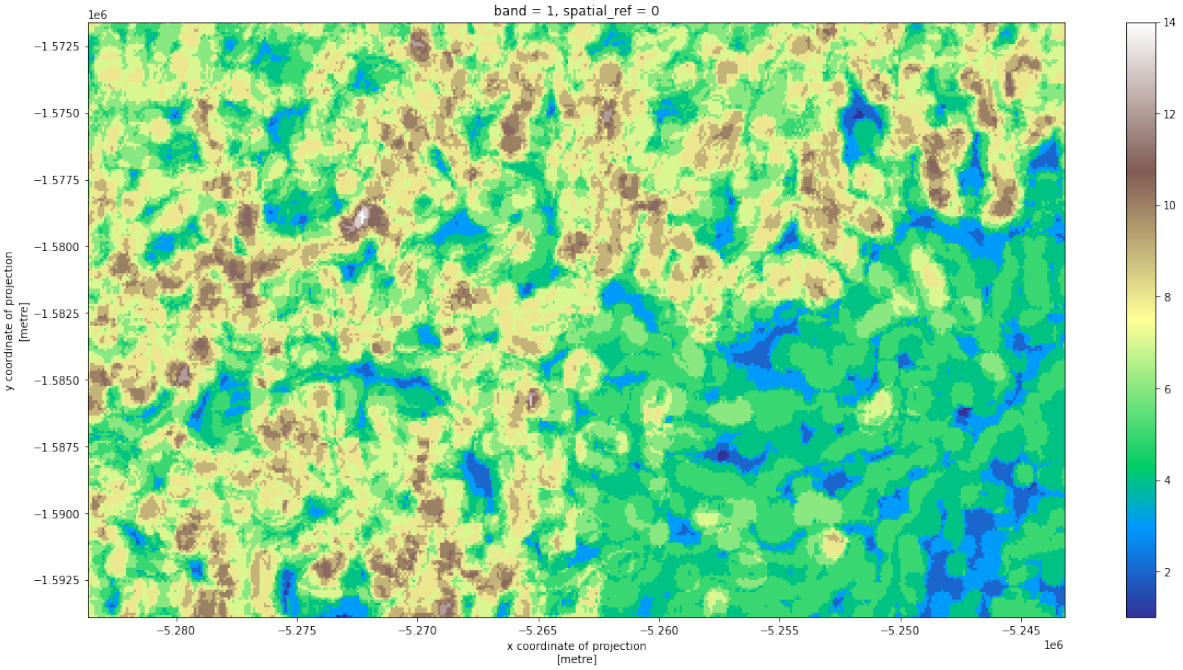

Figura 3. Variedade de landforms para região de Alto Paraíso de Goiás-GO, Brasil.

```
assetId = "projects/ee-lucasljardim9/assets/landform_variety"

geemap.ee_export_image_to_asset(
    landform_variety,
    description='landform_variety',
    assetId=assetId,
    region=bioma_box,
    scale=escala,maxPixels=1e13
)
```

Bibliografia

Anderson, M.G., Barnett, A., Clark, M., Ferree, C., Sheldon, A.O., Prince, J. 2016. Resilient Sites for Terrestrial Conservation in Eastern North America. The Nature Conservancy. http://easterndivision.s3.amazonaws.com/Resilient_Sites_for_Terrestrial_Conservation.pdf.

Farr, T.G., et al. 2007. The shuttle radar topography mission. Reviews of Geophysics, 45, 2, RG2004, https://doi.org/10.1029/2005RG000183.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., Moore, R. 2017. Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, 202, 18-27, https://doi.org/10.1016/j.rse.2017.06.031.

MapBiomas Project. 2020. Collection 7 of the Annual Series of Land Use and Land Cover Maps of Brazil. Accessed on 2023 through the link: projects/mapbiomas-workspace/public/collection7/mapbiomas_collection70_integration_v2.

Python Software Foundation. Python Language Reference. http://www.python.org.

Tadono, T., Ishida, H., Oda, F., Naito, S., Minakawa, K., Iwamoto, H. 2014. Precise Global DEM Generation by ALOS PRISM. Precise Global DEM Generation by ALOS PRISM. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, II-4, 71–76, https://doi.org/10.5194/isprsannals-ii-4-71-2014.

Theobald, D.M., Harrison-Atlas, D., Monahan, W.B., Albano, C.M. 2015. Ecologically-relevant maps of landforms and physiographic diversity for climate adaptation planning. Plos One, 12, 1-17, https://doi.org/10.1371/journal.pone.0143619.

Yamazaki, D., Ikeshima, D., Tawatari, R., Yamaguchi, T, O'Loughlin, F., Neal, J.C., Sampson, C.C., Kanae, S., Bates, P.D. 2017. A high-accuracy map of global terrain elevations. Geophysical Research Letters, 11, 5844-5853, doi:10.1002/2017GL072874.

Yamazaki, D., Ikeshima, D., Sosa, J., Bates, Paul, D., Allen, G.H., Pavelsky, T.M. 2019. MERIT Hydro: A High-Resolution Global Hydrography Map Based on Latest Topography Dataset. Water Resources Research, 6, 5053-5073, https://doi.org/10.1029/2019WR024873.

Iwahashi, J., Yamazaki, D. 2022. Global polygons for terrain classification divided into uniform slopes and basins. Progress in Earth and Planetary Science, 9, 33, https://doi.org/10.1186/s40645-022-00487-2.

Weiss, A.D., 2001. Topographic position and landforms analysis. Poster Presentation, ESRI Users Conference, San Diego, CA.

Wu, Q. 2020. geemap: A Python package for interactive mapping with Google. Journal of Open Source Software, 5, 51, 2305, https://doi.org/10.21105/joss.02305.

Amplitude de Elevação

Importando pacotes e inicializando geemap

```
import os
import ee
import geemap
```

```
geemap.ee_initialize()
```

A amplitude de elevação foi calculada como a diferença entre a elevação máxima e mínima dentro da vizinhaça da célula focal. Como essa métrica é correlacionada com a variedade de *landforms*, nós calculamos o resíduos de uma regressão (Ordinary Least Squares) entre as duas váriaveis. Assim, a amplitude de elevação residual é independente da variedade de *landforms*, permitindo a identificação de locais que tenham maior variabilidade microclimática que a proporcionada pela variedade de *landforms*, quando compormos o índice de diversidade da paisagem.

Base de Dados

Nós utilizamos o Modelo Digital de Elevação (DEM) do Merit-DEM (Yamazaki et al. 2017), na escala de 90 metros. O Merit-DEM é um produto global que combina dados dos satélites do Shuttle Radar Topography Mission (SRTM) (Farr et al. 2007) e Advanced Land Observing Satellite (ALOS) (Tadono et al. 2014), permitindo a replicabilidade da metodologia em outras regiões. Além disso, o Merit-DEM corrige viéses de Modelo Digitais de Elevação gerados por imagens de satétite como *speckle noise*, *stripe noise*, *absolute bias* e *tree height bias* (Yamazaki et al. 2017). A correção de *tree height bias* é principalmente importante para a Floresta Amazônica devido à sua densidade de árvores altas.

A variedade de *landforms* foi calculada anteriormente (veja o capítulo Variedade de *Landforms*) e está disponível com *asset* em **"projects/ee-lucasljardim9/assets/landform_variety"**.

As análises foram rodadas no *Google Earth Engine* (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote **geemap** (Wu 2020) em *Python* (Python Software Foundation 2023) como interface pela facilidade na documentação e reprodutividade das análises.

Códigos para o cálculo da amplitude de elevação residual

Nossas análises foram rodadas no Google Earth Engine (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote geemap (Wu 2020) em Python (Python Software Foundation 2023) como interface pela facilidade na documentação e reprodutividade das análises.

```
# Importando mapa de biomas do IBGE para extrair as coordenadas mínimas e máximas do Brasi
bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")

def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()
```

Nós importamos os *rasters* do modelo digital de elevação e da variedade de *landforms*.

```
# Importando o modelo digital de elevação
DEM = ee.Image("MERIT/DEM/v1_0_3")

# Importando a variedade de landforms calculada anteriormente
landform_variety = ee.Image("projects/ee-lucasljardim9/assets/landform_variety")

#Escala dos rasters ~92 metros
escala = DEM.projection().nominalScale()
```

Nós extraímos do *raster* as células dentro da vizinhança (kernel circular com 5 células de raio, ~450 metros) da célula focal e salvamos como bandas de uma imagem. Assim, cada banda é um *stack* das células da vizinhança da célula focal, a primeira banda possui todas as primeiras células de cada célula focal, a segunda banda todas as segunda células e assim por diante.

```
# Tamanho do raio do kernel para o calculo da amplitude de elevação
radius_pixels = 5

# Criando rasters da vizinhança de cada célula como bandas da imagem
neighbor = DEM.neighborhoodToBands(ee.Kernel.circle(ee.Number(radius_pixels)))
```

A imagem *neighbor* criada anteriormente as células da vizinhança como bandas da imagem. Assim, as primeiras células de cada banda são as células da vizinhança da primeira célula focal, organizadas como colunas (bandas). Ao calcularmos os valores máximos e mínimos para cada coluna de *neighbor*, estamos calculando os valores máximos e mínimos da vizinhança de cada célula focal.

```
# Calcule o máximo da vizinhança
elevation_max = neighbor.reduce(ee.Reducer.max())

# Calcule o mínimo da vizinhança
elevation_min = neighbor.reduce(ee.Reducer.min())
```

Subtraindo os valores máximos e mínimos de cada célula focal e calculando o seu valor absoluto, temos a amplitude de elevação para cada célula focal. Nós salvamos a amplitude de elevação e variedade de *landforms* como uma imagem com duas bandas, sendo a primeira banda a variável preditora da regressão e a segunda banda a variável resposta.

```
# Calcule a amplitude da vizinhança
elevation_range = elevation_max.subtract(elevation_min).abs()

# Crie uma imagem com as bandas de varidedade de landforms e amplitude de elevação
# A primeira imagem é o x da regressão e a segunda é o y

elevation = (ee.Image.cat(landform_variety, elevation_range)
            .rename(['landform_variety', 'elevation_range']))
```

Desta forma, aplicamos a regressão entre as variáveis.

```
# Rode uma regressão linear (OLS) entre varidade de landforms e amplitude de elevação
regression = elevation.reduceRegion(**{
     'reducer': ee.Reducer.linearFit(),
     'geometry': bioma_box,
     'maxPixels': 1e13,
     'scale': escala
    })
```

Após a regressão, multiplicamos a variável preditora pelo coeficiente de regressão (*slope*) e adicionamos o valor do intercepto para predizermos os valores de amplitude de elevação esperados pela regressão. Em seguida, subtraimos os valores de amplitude de elevação pelos valores preditos pela regressão para calcularmos os resíduos do modelo.

```
# Calcule o valor predito, pela regressão, de amplitude elevação, sem intercepto
pred = elevation.select('landform_variety').multiply(ee.Number(regression.get('scale')))

# Adicione o intercepto na predição
predict = pred.add(ee.Number(regression.get('offset')))

# Calcule o residuo da regressão
residuals = elevation.select('elevation_range').subtract(predict).rename(['residuals'])
```

Por fim, exportamos o raster de amplitude de elevação residual como um *asset* do Google Earth Engine.

```
# Exporte a amplitude de elevação residual como asset
assetId = "projects/ee-lucasljardim9/assets/elevetion_range_residual"

geemap.ee_export_image_to_asset(
    residuals,
    description='elevation_range_residual',
    assetId=assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)
```

Densidade e quantidade de áreas úmidas

Importando pacotes e inicializando *geemap*

```
import os
import ee
import geemap
```

```
geemap.ee_initialize()
```

O índice *Wetland score* é uma combinação da densidade de áreas úmidas localmente com a densidade e quantidade de áreas úmidas regionalmente. *Wetland score* entra no cálculo da diversidade da paisagem em locais planos e úmidos, com baixa variedade de *landforms* e baixa amplitude de elevação. Nesses locais a variação microclimática seria baixa devido a baixa variabilidade topográfica e geomorfológica, mas como há alta densidade de áreas úmidas, esses locais atuam regulando a variabilidade microclimática localmente, como tampões climáticos, e regulando a emissão de gases de efeito estufa.

Existem três cenários de distribuição de áreas úmidas, os locais (1) estão presentes em áreas com alta densidade de áreas úmidas no entorno, (2) os locais estão situados em áreas com baixa densidade de áreas úmidas localmente, mas alta densidade regionalmente e (3) os locais estão presentes em áreas com alta quantidade de áreas úmidas, mas baixa densidade devido a sua distribuição espacial. Desta forma, *wetland score* é composto pelos três cenários citados anteriormente, primeiro é calculado a densidade local, regional e a quantidade regional e, para cada métrica, é calculado um valor de Z, subtraindo pela média e dividindo pelo desvio padrão. A densidade de áreas úmidas é a média ponderada dos valores de Z da densidade local e regional (peso duplo para a densidade local). Nos locais onde os valores de Z da quantidade de áreas úmidas regional é maior que a densidade média calculada anteriormente, o índice torna-se a média ponderada da densidade local, densidade regional e quantidade de áreas úmidas regional (duplo peso para a densidade local).

$$wetland.score = \begin{cases} \frac{2 \times densidade_{local}(Z) + densidade_{regional}(Z)}{3} & \text{se densidade média é maior ou igual à quantidade regional} \\ \frac{2 \times densidade_{local}(Z) + densidade_{regional}(Z) + quantidade_{regional}(Z)}{4} & \text{se a quantidade regional for maior a densidade média} \end{cases}$$

Nesse capítulo demonstraremos como as densidades e a quantidade de áreas úmidas foram calculadas e em outro capítulo mostraremos como calculamos os valores de Z e o *wetland score*.

Banco de Dados

Nós utilizamos como base de dados de áreas úmidas o *Global Wetlands database* (Gumbricht et al. 2017). Nós reprojetamos o *raster* de áreas úmidas para a mesma resolução do modelo digital de elevação usado nas etapas anteriores (~ 90 metros).

Para calcularmos as densidades e quantidade de áreas úmidas, nós retiramos as áreas úmidas classificadas como sistemas lacustres e ribeirinhos (*riverines* e *lacustrines*) e reclassificamos o *raster* como sendo área úmida (1) ou não sendo área úmida (0). Depois, calculamos a densidade de áreas úmidas dentro de uma vizinhaça de 450 metros (5 células) de raio de um *kernel* circular (densidade local). Calculamos também a densidade e a quantidade de áreas úmidas na vizinhança de ~ 1170 metros (13 células) (regional).

Código para calcular as densidades e quantidade de áreas úmidas

As análises foram rodadas no *Google Earth Engine* (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote **geemap** (Wu 2020) em *Python* (Python Software Foundation 2023) como interface pela facilidade na documentação e reprodutividade das análises.

Primeiro, nós importamos os polígonos dos biomas do Brasil e extraímos suas coordenadas geográficas máximas e mínimas para delimitar a região de análise. Importamos o modelo digital de elevação e o *raster* de áreas úmidas e reprojetamos a resolução das áreas úmidas para a resolução do modelo digital de elevação.

```
# Importando mapa de biomas do IBGE para extrair as coordenadas
# mínimas e máximas do Brasil
bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")

def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()

# Extraindo a resolução do mapa
DEM = ee.Image("MERIT/DEM/v1_0_3")

escala = DEM.projection().nominalScale()
```

```
# Reprojetando áreas úmidas
wetlands = (ee.Image("projects/ee-lucasljardim9/assets/Cifor_wetlands")
            .reproject(**{'crs': "EPSG:4326",
                          'scale': escala}))
```

Em seguida, retiramos os sistemas ribeirinhos e lacustres do *raster* de áreas úmidas e reclassificamos as classes do raster em presença e ausência de áreas úmidas.

```
# Criando uma máscara para rios e lagos

rivers = wetlands.mask(wetlands.neq(10))

# Retirando rios e lagos das áreas úmidas

wetlands = wetlands.mask(rivers)

# Transformando áreas úmidas em um raster binário
# de presença de áreas úmidas

wetlands_binary = wetlands.where(wetlands.gt(0), 1)
```

Posteriomente, nós calculamos a densidade de áreas úmidas, localmente, dentro de um *kernel* circular de ~450 metros (5 células). Primeiro, transformamos as células da vizinhança de cada célula focal em bandas de uma imagem. Assim, cada células vizinha da célula focal fica empilhada como uma coluna. Para cada coluna, somamos os valores das células (0 ou 1) como a quantidade de áreas úmidas na vizinhança. Depois, dividimos a quantidade de áreas úmidas pelo número de células na vizinhança, resultando na densidade de áreas úmidas.

```
radius_pixels = 5

# Tranforme as células da vizinha em bandas

neighbors = wetlands_binary
            .neighborhoodToBands(ee.Kernel.circle(ee.Number(radius_pixels)))

# Conte a quantidade de áreas úmidas na vizinhança
wetlands_count = neighbors.reduce(ee.Reducer.sum()).toDouble()

# Conte o número de células totais na vizinhança

neighbors_amount = neighbors.bandNames().length()
```

```
#Divida a quantidade de áreas úmidas pelo
# número de células para calcular a densidade
wetlands_density_local = wetlands_count.divide(ee.Number(neighbors_amount))
```

Repetimos o mesmo procedimento da densidade local para calcularmos a densidade e quantidade de áreas úmidas regional.

```
radius_pixels = 13

# Transforme as células vizinhas em bandas

neighbors = wetlands_binary
           .neighborhoodToBands(ee.Kernel.circle(
                                    ee.Number(radius_pixels)
                                    )
                                )

# Conte a quantidade de áreas úmidas
wetlands_count = neighbors.reduce(ee.Reducer.sum()).toDouble()

# Conte o número de células totais na vizinhança
neighbors_amount = neighbors.bandNames().length()

# Calcule a densidade dividindo a quantidade
# de áreas úmidas pelo número de células
wetlands_density_regional = wetlands_count.divide(ee.Number(neighbors_amount))
```

Por fim, exportamos a densidade de áreas úmidas local (*wetlands_density*), a densidade regional (*wetlands_density_1000*) e a quantidade regional (*wetlands_count*) como *assets* no *Google Earth Engine*.

```
assetId_quantidade = "projects/ee-lucasljardim9/assets/wetlands_count"

assetId_densidade_local = "projects/ee-lucasljardim9/assets/wetlands_density"

assetId_densidade_regional = "projects/ee-lucasljardim9/assets/wetlands_density_1000"

geemap.ee_export_image_to_asset(
    wetlands_count,
    description='wetlands_count',
    assetId=assetId_quantidade,
```

```
    region=bioma_box,
    scale=escala, maxPixels=1e13
)

geemap.ee_export_image_to_asset(
    wetlands_density_local,
    description='wetlands_density_local',
    assetId=assetId_densidade_local,
    region=bioma_box,
    scale=escala, maxPixels=1e13
)

geemap.ee_export_image_to_asset(
    wetlands_density_regional,
    description='wetlands_density_regional',
    assetId=assetId_densidade_regional,
    region=bioma_box,
    scale=escala, maxPixels=1e13
)
```

Diversidade de solo

Importando pacotes e inicializando *geemap*

```
import os
import ee
import geemap
```

```
geemap.ee_initialize()
```

A diversidade de solo é incluida na diversidade da paisagem em locais com baixa variedade de *landforms*, amplitude de elevação e *wetland score*. A diversidade de solos é calculada somando o número de tipos de solo presentes na vizinhaça da célula focal.

Banda de Dados

Nós utilizamos a base de dados de pedologia do Instituto Brasileiro de Geografia e Estatística (IBGE) como base para o cálculo da diversidade de solos. Os polígonos foram posteriomente filtrados por tipo de solo e rasterizados na escala do modelo digital de elevação que usamos nas etapas anteriores (~ 90 metros).

Códigos para a criação da diversidade de solos

As análises foram rodadas no *Google Earth Engine* (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote **geemap** (Wu 2020), em *Python* (Python Software Foundation 2023), como interface pela facilidade na documentação e reprodutividade das análises.

O primeiro passo é criar uma função que filtra os polígonos pelo nome do solo, atribui valor 1 ao polígono, transforma em *raster* e reprojeta para a resolução do modelo digital de elevação que estamos usando em todas as análises (~90 metros). Depois, a função cria uma imagem cujas bandas são as células da vizinhaça da célula focal. Células com a presença do solo tem

valor 1 e sem o solo 0. Ao somarmos os valores das células dentro da vizinhança temos o número de células com o tipo de solo. Por fim, transformamos valores maior que 0 em 1 para classificarmos o *raster* como presença ou ausência do tipo de solo na vizinhança da célula focal.

```
# Função para mudar o valor da propriedade do polígono para 1
def set_feature(feature):
               return feature.set("cod_simbol", 1)

# Função para filtrar o polígono por tipo de solo,
# transformar em raster e retornar um raster de presença
# e ausência do solo na vizinhança
def soil_presence(code):
    # Filtra  o tipo de solo e atribui valor 1 (para rasterizar)
    type = (soil.filter(ee.Filter.eq("cod_simbol", code))
           .map(set_feature))

    # Transforma o polígono em raster
    soil_rast = type.reduceToImage(**{
                      'properties': ["cod_simbol"],
                      'reducer': ee.Reducer.first()
                    })

    # Converte o raster para a escala do modelo digital de elevação
    soil_raster = (soil_rast
                  .reproject(**{'crs': "EPSG:4326",
                                'scale': escala}))

    radius_pixels = 15

    # Conte o número de células na vizinhança que possuem o tipo de solo selecionado
    soil_count = (soil_raster.neighborhoodToBands(ee.Kernel.circle(radius_pixels))
                  .reduce(ee.Reducer.count()))

    # Tranforme os valore maiores que 0 em 1, indicando a presença
    #ou ausência do tipo de solo na vizinhança
    soil_diversity = soil_count.where(soil_count.gt(0), 1)

    return soil_diversity
```

```
# Importando mapa de biomas do IBGE para extrair as coordenadas mínimas e máximas do Brasi
bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")

def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()

# Extraindo a resolução do mapa
DEM = ee.Image("MERIT/DEM/v1_0_3")

escala = DEM.projection().nominalScale()
```

Nós importamos os polígonos de solo do IBGE e filtramos a coluna com o tipo de solo ("cod_simbol"), e criamos uma lista com os nomes dos tipos de solo, eliminando os polígonos que não tinham nomes.

```
# Importando os polígonos de solo do IBGE e selecionado a coluna com tipos de solo
soil = (ee.FeatureCollection("projects/ee-lucasljardim9/assets/soil_IBGE")
        .select("cod_simbol"))

# Criando uma lista com os nomes dos tipos de solo
soil_list = soil.reduceColumns(ee.Reducer.toList(), ["cod_simbol"]).values().get(0)

# Criando uma lista com os nomes de solo, sem duplicatas
soil_codes = (ee.List(soil_list).distinct()
              .filter(ee.Filter.neq("item", "")))
```

Depois, aplicamos a função que criamos anteriormente para cada nome de solo e transformamos em bandas de uma imagem. Ao somarmos os valores de cada célula nas bandas calculamos o ńumero de tipos de solo presentes na vizinhança de cada célula focal, a diversidade de solos.

```
# Crie o raster de presença e ausencia para cada tipo de solo
soil_list = soil_codes.map(soil_presence)

# Transforme os raster de presença de solo em bandas de
# uma imagem e soma para calcular a diversidade de solos
# Converta do doble para permitir a exportação do dado
soil_diversity = (ee.ImageCollection.fromImages(soil_list)
                  .toBands()
                  .reduce("sum")
```

```
                  .toDouble())
```

```
# Exporte o raster de diversidade de solo
assetId = "projects/ee-lucasljardim9/assets/soil_diversity"

geemap.ee_export_image_to_asset(
    soil_diversity, description='soil_diversity', assetId=assetId, region=bioma_box, scale
)
```

Calculando *Z-scores* das variáveis

```
import os
import ee
import geemap
```

```
geemap.ee_initialize()
```

Antes de calcularmos a diversidade da paisagem e a resiliência, nós transformamos as variáveis, variedade de *landforms*, amplitude de elevação, densidade e quantidade de áreas úmidas e diversidade de solo, em valores de Z. O cálculo de Z é realizado dentro de regiões similares ecologicamente e geologicamente (unidade eco-geológicas), criadas anteriormente. Além disso, os valores de densidade e quantidade de áreas úmidas são combinados em *wetland score*.

Os valores de Z de cada variável, em cada célula (i), é o desvio do valor da célula da média da unidade eco-geológica (u), divido pelo seu desvio padrão:

$$Z - score_{iu} = \frac{i - media_u}{desviopadrao_u}$$

O *wetland score* é a média ponderada das densidades locais e regionais de áreas úmidas. Em locais onde o valor de Z da densidade média é menor que o Z da quantidade de áreas úmidas, o *wetland score* assume o segundo valor Z.

$$wetland.score = \begin{cases} \frac{2 \times densidade_{local}(Z) + densidade_{regional}(Z)}{3} & \text{se densidade média é maior ou igual à quantidade regional} \\ \frac{2 \times densidade_{local}(Z) + densidade_{regional}(Z) + quantidade_{regional}(Z)}{4} & \text{se a quantidade regional for maior a densidade média} \end{cases}$$

Base de Dados

Nós utilizamos as variáveis criadas anteriormente e guardadas como *assets* no *Google Earth Engine*. As variáveis são:

- variedade de *landforms*

- amplitude de elevação
- diversidade de solo
- unidades eco-geológicas
- conectividade da paisagem
- densidade local de áreas úmidas
- densidade regional de áreas úmidas
- quantidade regional de áreas úmidas

Códigos para o cálculo dos *Z-scores*

Primeiro, nós criamos uma função para calcular o *Z-score* dentro de cada unidade eco-geológica (**code**) para cada variável (**image**). A função cria uma máscara da variável pelo código da unidade eco-geológica e calcula as médias e desvio padrão de cada unidade, depois as variáveis originais são subtraídas da média de sua unidade e divida pelo desvio padrão.

```
def calculate_Z(code, image):
  #Crie uma máscara de unidade eco-geologica para cada codigo da unidade (code)
  mask = (geophysical_setting
          .where(geophysical_setting.eq(ee.Number.parse(code)), 1)
          .where(geophysical_setting.neq(ee.Number.parse(code)), 0))

  # Corte a imagem pela máscara
  map1 = image.mask(mask).rename("b1")
  # Calcule a média de cada unidade
  mean = (map1.reduceRegion(**{'reducer':ee.Reducer.mean(),
                               'geometry':bioma_box,
                               'scale':escala,
                               'maxPixels':1e13
                              })
             .get("b1"))
  # Calcule o desvio padrão de cada unidade
  sd = (map1.reduceRegion(**{'reducer':ee.Reducer.stdDev(),
                             'geometry':bioma_box,
                             'scale':escala,
                             'maxPixels':1e13
                            })
            .get("b1"))

  # calcule o Z
  Z = map1.subtract(ee.Number(mean)).divide(ee.Number(sd))
```

```
  return Z
```

Em seguida, criamos também um função para aplicar o **calculate_Z** para uma imagem definida pelo usuário transforme em uma image de banda única.

```
def wrap_calculate_Z(image):
    # Aplique calculate_Z para cada unidade (classe), guarde numa ImageCollection,
    # transform em bandas e resuma numa imagem de banda única
    Z_map = (ee.ImageCollection(classes.map(lambda i: calculate_Z(i, image)))
             .toBands()
             .reduce("sum"))

    return Z_map
```

Aqui nós importamos as variáveis de interesse para o cálculo de Z.

```
# Importando raster das unidades eco-geologicas

geophysical_setting = (ee.Image("projects/ee-lucasljardim9/assets/ecoregions_geology")
                       .unmask())

# Importando as variáveis

landform_variety = ee.Image("projects/ee-lucasljardim9/assets/landform_variety")

elevation_range = ee.Image("projects/ee-lucasljardim9/assets/elevation_range_residual")

soil_diversity = ee.Image('projects/ee-resilience/assets/New_window_size/soil_diversity')

connectedness = ee.Image("projects/ee-lucasljardim9/assets/Biomas_resistencia_kernel")

# Importando os dados de wetlands para o wetland score

wetlands_count = ee.Image("projects/ee-lucasljardim9/assets/wetlands_count")

wetlands_density = ee.Image("projects/ee-lucasljardim9/assets/wetlands_density")

wetlands_density_1000 = ee.Image("projects/ee-lucasljardim9/assets/wetlands_density_1000")

# Importando o polígono de bioma para definir as
# coordenadas máximas e mínimas do Brasil
```

```
bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")

# ModeloDigital de Elevação para extrair a resolução
DEM = ee.Image("MERIT/DEM/v1_0_3")

# função para extrair as bordas dos polígonos
def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()

# Resolução das análises
escala = DEM.projection().nominalScale()
```

Nós criamos uma lista com os nomes das unidades eco-geológicas (classes).

```
# extraindo os valores do raster de unidades eco-geológicas em histograma
histogram = geophysical_setting.reduceRegion(**{'reducer':ee.Reducer.frequencyHistogram(),
                                                'geometry': bioma_box,
                                                'scale': escala,
                                                'maxPixels':1e13
                                            })
# Criando uma lista com os nomes das unidades eco-geológicas
classes = ee.Dictionary(histogram.get("b1")).keys()
```

Calculando Z-scores

```
Nós aplicamos a função **wrap_calculate_Z** para cada variável e guardamos o valores de Z
```

```
# Calculando os valores de Z para cada variável
Z_landform_variety = wrap_calculate_Z(landform_variety)

Z_elevation_range = wrap_calculate_Z(elevation_range)

Z_soil_diversity = wrap_calculate_Z(soil_diversity)

Z_wetlands_count = wrap_calculate_Z(wetlands_count)

Z_wetlands_density = wrap_calculate_Z(wetlands_density)
```

```
Z_wetlands_density_1000 = wrap_calculate_Z(wetlands_density_1000)

Z_connectedness = wrap_calculate_Z(connectedness)
```

Calculando wetland score

Aqui, nós calculamos o *wetland score*, aplicando a fórmula descrita anteriormente.

```
# Calculando a densidade de areas úmidas como a média do local e regional
wetlands_density = (Z_wetlands_density
                    .multiply(2)
                    .add(Z_wetlands_density_1000)
                    .divide(3))

# Testando se o Z da quantidade de áreas úmidas é maior que a densidade média

wet_test = wetlands_density.lt(Z_wetlands_count)

# Média de densidade local, regional e quantidade de áreas úmidas
wet_average = (wetlands_density
               .multiply(3)
               .add(Z_wetlands_count)
               .divide(4))

# Substituindo os locais com densidade menor que a quantidade pelos valores de quantidade
Z_wetlands_score = Z_wetlands_density.where(wet_test, wet_average)
```

Exportando os Z-scores

Por último, exportamos todas as imagens de Z como *asset* no *Google Earth Engine*.

```
# Criando os links dos assets
landform_assetId = "projects/ee-lucasljardim9/assets/Z_landform_variety_byregion"

elevation_assetId = "projects/ee-lucasljardim9/assets/Z_elevation_range_region"

wetland_assetId = "projects/ee-lucasljardim9/assets/Z_wetlands_score_byregion"

soil_assetId = "projects/ee-lucasljardim9/assets/Z_soil_diversity_byregion"
```

```
connectedness_assetId = "projects/ee-lucasljardim9/assets/Z_connectedness_byregion"

# Exportando as imagens
geemap.ee_export_image_to_asset(
    Z_landform_variety,
    description='Z_landform_variety',
    assetId=landform_assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)

geemap.ee_export_image_to_asset(
    Z_elevation_range,
    description='Z_elevation_range',
    assetId=elevation_assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)

geemap.ee_export_image_to_asset(
    Z_wetlands_score,
    description='Z_wetlands_score',
    assetId=wetland_assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)

geemap.ee_export_image_to_asset(
    Z_soil_diversity,
    description='Z_soil_diversity',
    assetId=soil_assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)

geemap.ee_export_image_to_asset(
    Z_connectedness,
```

```
    description='Z_connectedness',
    assetId=connectedness_assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)
```

Diversidade da paisagem

Importando pacotes e inicializando *geemap*

```
import os
import ee
import geemap
```

```
geemap.ee_initialize()
```

A diversidade da paisagem é uma métrica composta pelos valores de Z da variedade de *landforms*, amplitude de elevação, *wetland score* e diversidade de solos. Inicialmente, a diversidade da paisage assume os valores de variedade de *landforms*. Nos locais onde a amplitude de elevação é maior que variedade de *landforms*, a diversidade da paisagem é substituída pela média ponderada entre as duas variáveis, atribuindo peso dobrado para a variedade de *landforms*. Locais onde wetland score é maior que a diversidade da paisagem calculada anteriormente, os valores são substituídos por wetland score. Na localidades onde a diversidade de solo é maior que a diversidade da paisagem, os valores são substituídos pela média ponderada das variáveis naquela localidade. A última etapa é truncar os valores de Z que são *outliers* para deixar a distribuição de diversidade da paisagem mais suavizada. A suavização é importante para que locais com alta diversidade da paisagem, devido a sua especificidade de variedade de landforms e amplitude da elevação não possuam mais peso no mapa de resiliência que os locais guiados por *wetland score*.

Base de Dados

Os dados para a diversidade da paisagem são os criados na etapa de cálculo dos *Z-scores*, descrito anteriormente.

Códigos para a criação da diversidade da paisagem

Nossas análises foram rodadas no Google Earth Engine (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote geemap (Wu 2020) em Python (Python Software Foundation 2023) como interface pela facilidade na documentação e reprodutividade das análises.

O primeiro passo é criar uma função para truncar a diversidade da paisagem por uma porcentagem (**coverage**) de cobertura da distribuição dos valores.

```
def truncate_z_scores(image, coverage):

    # Calcule os percentis baseado na cobertura definida em coverage
    percentile = [100 - coverage, coverage]

    # Encontre os valores dos percentis definidos
    quartiles = image.reduceRegion(**{'reducer': ee.Reducer.percentile(percentile),
                                      'scale': image.projection().nominalScale(),
                                      'maxPixels':1e13});
    #5% quantiles
    q5 = quartiles.getNumber('sum_p5')
    #95% quantiles
    q95 = quartiles.getNumber('sum_p95')

    # Truncar a imagem pelo percentis
    truncated_z = (image.where(image.lt(q5), q5)
                   .where(image.gt(q95), q95))

    return truncated_z
```

Nós importamos os dados de bioma para delimitar a região de análise e o raster do Brasil para que os mapas sejam cortados para o Brasil antes do cálculo da diversidadade da paisagem.

```
# Importando o polígono de bioma para definir as
# coordenadas máximas e mínimas do Brasil

bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")

brasil_raster = ee.Image("projects/ee-lucasljardim9/assets/brasil_raster")

# ModeloDigital de Elevação para extrair a resolução
```

```
DEM = ee.Image("MERIT/DEM/v1_0_3")

# função para extrair as bordas dos polígonos
def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()

# Resolução das análises
escala = DEM.projection().nominalScale()
```

Importamos os mapas e cortamos para o raster do Brasil.

```
# Importando e cortando as imagens para o Brasil
landform_Z = (ee.Image('projects/ee-lucasljardim9/assets/Z_landform_variety_byregion')
              .updateMask(brasil_raster))

elevation_range_Z = (ee.Image('projects/ee-lucasljardim9/assets/Z_elevation_range_byregion
                     .updateMask(brasil_raster))

wetland_score = (ee.Image('projects/ee-lucasljardim9/assets/Z_wetlands_score_byregion')
                 .updateMask(brasil_raster))

soil_diversity =  (ee.Image('projects/ee-lucasljardim9/assets/Z_soil_diversity_byregion')
                   .updateMask(brasil_raster))
```

Aqui começamos o cálculo da diversidade da paisagem, atribuindo à diversidade da paisagem os valores de variedade de *landforms*. Depois testamos os locais onde a amplitude de elevação é maior que a variedade de *landforms* e substituímos os valores.

```
landscape_diversity = landform_Z

# Testando se o Z da amplitude de elevação é maior que o Z da variedade de landsforms
test_1 = elevation_range_Z.gt(landscape_diversity)

# Média ponderada entre variedade de landforms e amplitude de elevação
average_elevation = landform_Z.multiply(2).add(elevation_range_Z).divide(3)

# Substituindo os valores de variedade de landforms por amplitude de elevação
# onde o segundo valor é maior que o primeiro
```

```
landscape_diversity = landscape_diversity.where(test_1, average_elevation)
```

Agora, nós adicionamos *wetland score*, primeiro testando onde *wetland score* é maior que a diversidade da paisagem calculada anteriormente, e nos locais cujos valores de *wetland score* são maiores que a diversidade da paisagem, nós substituímos os valores.

```
# testando onde wetland score é maior que a diversidade da paisagem
# com variedade de landforms e amplitude de elevação
test_2 = wetland_score.gt(landscape_diversity)

# Substitua o valor de diversidade da paisagem por wetland score
#onde o segundo é maior que o primeiro
landscape_diversity = landscape_diversity.where(test_2, wetland_score)
```

A inclusão da diversidade de solos é um pouco mais complexa. Nós testamos onde a diversidade de solo é maior a diversidade da paisagem já calculada e substituímos os valores por 4 médias ponderadas diferentes, cada uma representando os passos anteriores. A primeira média é para os locais onde a diversidade de solos é maior que *wetland score*, que é maior que amplitude de elevação. Na segunda média, os valores são para as regiões com maior diversidade de solos, mas onde *wetland score* é menor que a amplitude de elevação. A terceira média é para locais onde a amplitude de elevação não é maior que variedade de *landforms*, mas possui valores menores de *wetland score*, que também são menores que a diversidade de solos. A quarta média é para locais onde somente a diversidade de solos é maior que a variedade de *landforms*.

```
# teste onde diversidade de solo é maior que a diversidade da paisagem
test_3 = soil_diversity.gt(landscape_diversity)

# Calcule as médias ponderadas
average_soil_1 = (landform_Z.multiply(2)
                  .add(elevation_range_Z)
                  .add(wetland_score)
                  .add(soil_diversity)
                  .divide(5))

average_soil_2 = (landform_Z.multiply(2)
                  .add(elevation_range_Z)
                  .add(soil_diversity)
                  .divide(4))

average_soil_3 = (landform_Z.multiply(2)
                  .add(wetland_score)
```

```
                  .add(soil_diversity)
                  .divide(4))

average_soil_4 = (landform_Z.multiply(2)
                  .add(soil_diversity)
                  .divide(3))

# Substitua os valores de diversidade da paisagem pela diversidade de solo
# os os valores de solo são maiores
landscape_diversity = landscape_diversity
                      .where(test_1.And(test_2).And(test_3), average_soil_1)

landscape_diversity = landscape_diversity
                      .where(test_1.And(test_2.Not()).And(test_3), average_soil_2)

landscape_diversity = landscape_diversity
                      .where(test_1.Not().And(test_2).And(test_3), average_soil_3)

landscape_diversity = landscape_diversity
                      .where(test_1.Not().And(test_2.Not()).And(test_3), average_soil_4)
```

Com o mapa de diversidade da paisagem pronto, nos aplicamos um truncamento dos valores maiores e menores que 95% da distribuição dos dados, para que haja uma suavização da imagem, impedindo que *outliers* guiem os mapas posteriores de resiliência da paisagem.

```
truncated_landscape_diversity = truncate_z_scores(landscape_diversity, 95)
```

Finalizamos exportando o mapa de diversidade da paisagem como *asset* no *Google Earth Engine*.

```
# Exporte o raster de diversidade da paisagem
assetId = "projects/ee-lucasljardim9/assets/landscape_diversity_byregion"

geemap.ee_export_image_to_asset(
    truncated_landscape_diversity,
    description='landscape_diversity_byregion',
    assetId=assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)
```

Resiliência da Paisagem

Importando pacotes e inicializando *geemap*

```
import os
import ee
import geemap
```

```
geemap.ee_initialize()
```

```
A resiliência da paisagem é a média da diversidade e connectividade da paisagem.
```

Base de Dados

Os dados para a resiliência da paisagem são o mapa de diversidade da paisagem e *Z-score* da connectividade criados anteriormente.

Códigos para a criação da resiliência da paisagem

As análises foram rodadas no *Google Earth Engine* (Gorelick 2017), devido a demanda computacional do projeto, usando o pacote **geemap** (Wu 2020) em *Python* (Python Software Foundation 2023) como interface pela facilidade na documentação e reprodutividade das análises.

Primeiro, importamos o polígono de biomas do Brasil para extraímos as coordenadas geográficas mínimas e máximas. Extraímos também a escala do modelo digital de elevação que estamos usando na análises prévias.

```
# Importando o polígono de bioma para definir as
# coordenadas máximas e mínimas do Brasil

bioma = ee.FeatureCollection("projects/ee-lucasljardim9/assets/Biome")
```

```
# ModeloDigital de Elevação para extrair a resolução
DEM = ee.Image("MERIT/DEM/v1_0_3")

# função para extrair as bordas dos polígonos
def func_cmp(feature):
    return feature.bounds()

# Extraindo as coordenadas mínimas e máximas do Brasil
bioma_box = bioma.map(func_cmp).geometry().dissolve(**{'maxError': 1}).bounds()

# Resolução das análises
escala = DEM.projection().nominalScale()
```

Importamos os mapas de diversidade da paisagem e *Z-score* da connectividade e calculamos a média entre eles.

```
Z_connectedness = ee.Image("projects/ee-lucasljardim9/assets/Z_connectedness_byregion")

Z_landscape_diversity = ee.Image("projects/ee-lucasljardim9/assets/landscape_diversity_byr

resilience = Z_connectedness.add(Z_landscape_diversity).divide(2)
```

Exportamos o mapa de resiliência como um *asset* no *Google Earth Engine*.

```
# Exporte o raster de diversidade da paisagem
assetId = "projects/ee-lucasljardim9/assets/resilience_byregion"

geemap.ee_export_image_to_asset(
    resilience,
    description='resilience_byregion',
    assetId=assetId,
    region=bioma_box,
    scale=escala,
    maxPixels=1e13
)
```