```
%pip install numpy pyscipopt matplotlib
```

```
Requirement already satisfied: numpy in e:\documentos\ufrj\10 periodo\
otm\otimizacao-rateio\.venv\lib\site-packages (2.3.5)
Requirement already satisfied: pyscipopt in e:\documentos\ufrj\10
periodo\otm\otimizacao-rateio\.venv\lib\site-packages (6.0.0)
Requirement already satisfied: matplotlib in e:\documentos\ufrj\10
periodo\otm\otimizacao-rateio\.venv\lib\site-packages (3.10.7)
Requirement already satisfied: contourpy>=1.0.1 in e:\documentos\ufrj\
10 periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in e:\documentos\ufrj\10
periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in e:\documentos\
ufrj\10 periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (4.61.0)
Requirement already satisfied: kiwisolver>=1.3.1 in e:\documentos\
ufrj\10 periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in e:\documentos\ufrj\
10 periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in e:\documentos\ufrj\10
periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (12.0.0)
Requirement already satisfied: pyparsing>=3 in e:\documentos\ufrj\10
periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in e:\documentos\
ufrj\10 periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in e:\documentos\ufrj\10
periodo\otm\otimizacao-rateio\.venv\lib\site-packages (from python-
dateutil>=2.7->matplotlib) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

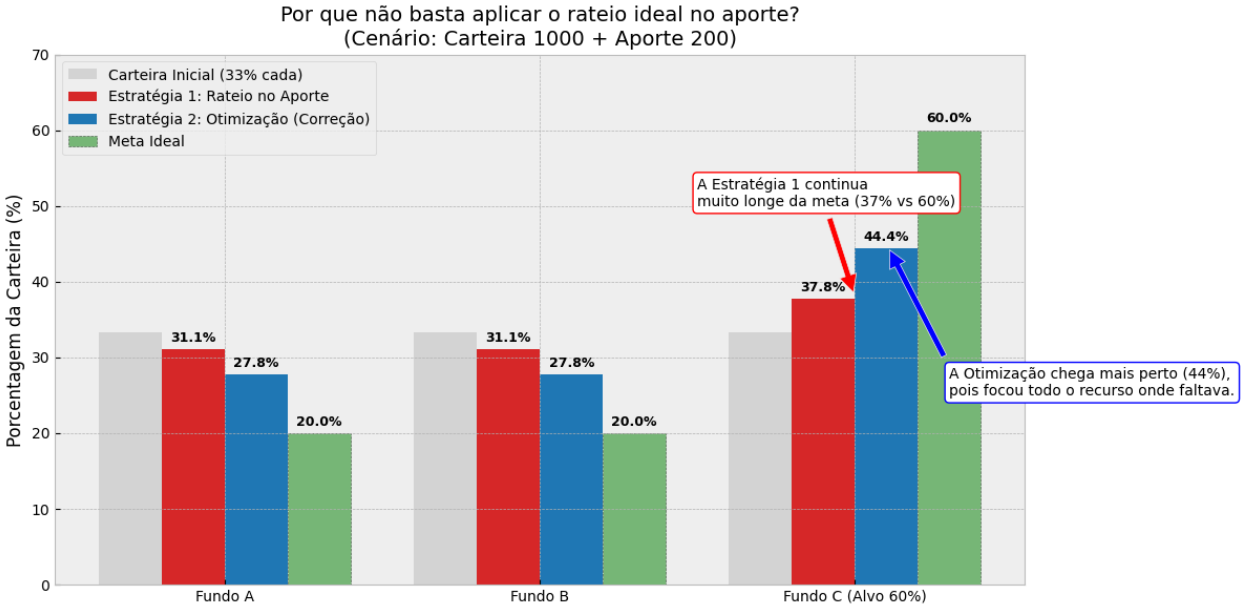
```
import numpy as np
import numpy.typing as npt
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from pyscipopt import Model, quicksum

plt.style.use('bmh')
```

Minimizar o erro de alocações de trades entre fundos

```
def plot_aporte_flow_vs_stock():
    # 1. Configuração do Cenário
    funds = ['Fundo A', 'Fundo B', 'Fundo C (Alvo 60%)']

    # Estado Inicial (Total 1000)
    initial_pos = np.array([333.33, 333.33, 333.33])
    initial_total = initial_pos.sum()

    # Metas (Target)
    target_ratios = np.array([0.20, 0.20, 0.60])

    # Aporte
    aporte = 200
    new_total = initial_total + aporte

    # --- ESTRATÉGIA 1: Aporte "Ingênuo" (Flow-based) ---
    # Aplica a % ideal apenas no dinheiro novo
    trades_naive = aporte * target_ratios # [40, 40, 120]
    final_pos_naive = initial_pos + trades_naive
    final_ratios_naive = final_pos_naive / new_total

    # --- ESTRATÉGIA 2: Aporte "Otimizado" (Stock-based) ---
    # Tenta corrigir o estoque.
    # O Fundo C precisa chegar a 720 (60% de 1200). Ele tem 333. Falta
muito.
    # Os Fundos A e B precisam ser 240 (20% de 1200). Eles tem 333.
Estão sobrando.
    # Logo, a otimização coloca TUDO no Fundo C para tentar diminuir o
erro.
    trades_opt = np.array([0.0, 0.0, 200.0])
    final_pos_opt = initial_pos + trades_opt
    final_ratios_opt = final_pos_opt / new_total

    # --- PLOTAGEM ---
    x = np.arange(len(funds))
    width = 0.2

    fig, ax = plt.subplots(figsize=(12, 6))

    # Barras
    rects1 = ax.bar(x - 1.5*width, initial_pos/initial_total*100,
width, label='Carteira Inicial (33% cada)', color='lightgray')
    rects2 = ax.bar(x - 0.5*width, final_ratios_naive*100, width,
label='Estratégia 1: Rateio no Aporte', color='tab:red')
    rects3 = ax.bar(x + 0.5*width, final_ratios_opt*100, width,
```

```
label='Estratégia 2: Otimização (Correção)', color='tab:blue')
    rects4 = ax.bar(x + 1.5*width, target_ratios*100, width,
label='Meta Ideal', color='green', alpha=0.5, linestyle='--',
edgecolor='black')

    # Configurações do Gráfico
    ax.set_ylabel('Porcentagem da Carteira (%)')
    ax.set_title(f'Por que não basta aplicar o rateio ideal no
aporte?\n(Cenário: Carteira 1000 + Aporte 200)', fontsize=14)
    ax.set_xticks(x)
    ax.set_xticklabels(funds)
    ax.legend()

    ax.set_ylim(0, 70) # Dá espaço para as labels

    # Adicionando Rótulos de Valor
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate(f'{height:.1f}%',
                        xy=(rect.get_x() + rect.get_width() / 2,
height),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom', fontsize=9,
fontweight='bold')

    autolabel(rects2)
    autolabel(rects3)
    autolabel(rects4)

    # Anotação Explicativa
    plt.annotate(
        "A Estratégia 1 continua\nmuito longe da meta (37% vs 60%)",
        xy=(2, 38), xytext=(1.5, 50),
        arrowprops=dict(facecolor='red', shrink=0.05),
        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="red",
lw=1)
    )

    plt.annotate(
        "A Otimização chega mais perto (44%),\npois focou todo o
recurso onde faltava.",
        xy=(2.1, 45), xytext=(2.3, 25),
        arrowprops=dict(facecolor='blue', shrink=0.05),
        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="blue",
lw=1)
    )

    plt.tight_layout()
```

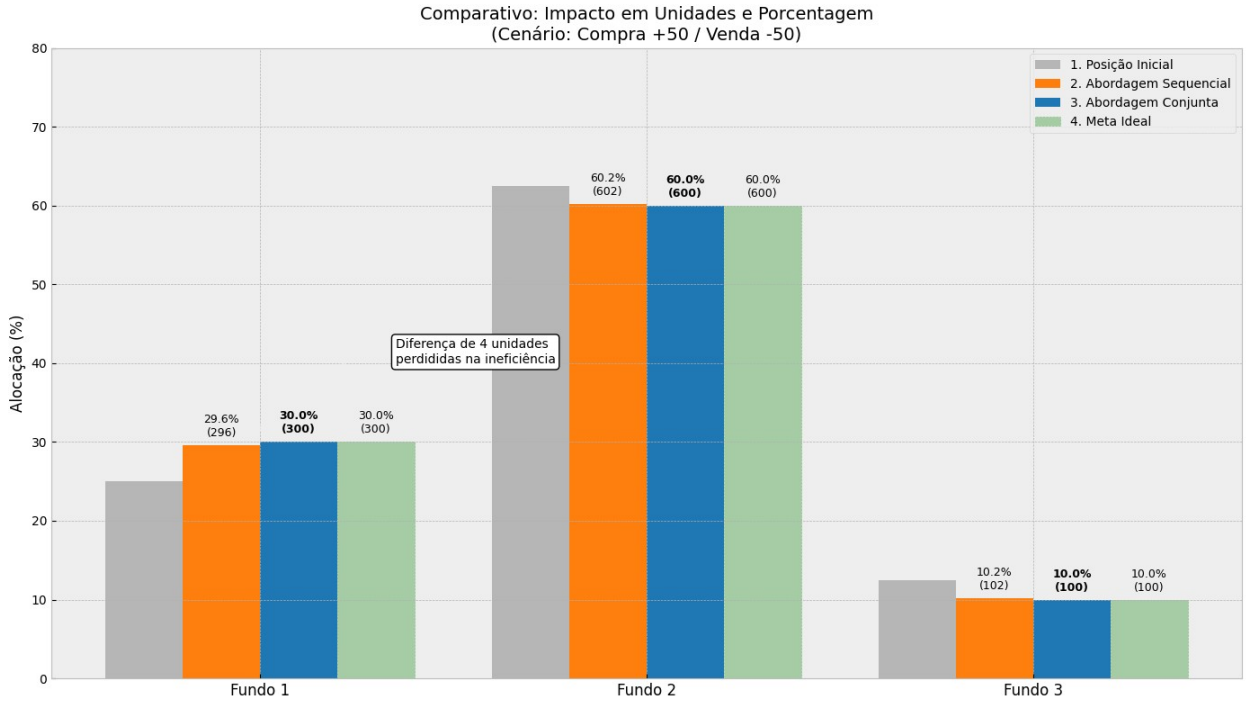
```
    plt.show()

plot_aporte_flow_vs_stock()
```




```
import matplotlib.pyplot as plt
import numpy as np

def plot_comparacao_estrategias_com_unidades_v2():
    # --- 1. Dados de Entrada (Escalonados / 10) ---
    funds = ['Fundo 1', 'Fundo 2', 'Fundo 3']

    # Posição Inicial: 250, 625, 125 (Total 1000)
    initial_pos = np.array([250, 625, 125])
    total_assets = initial_pos.sum() # 1.000

    # Metas (Target): 30%, 60%, 10%
    ideal_ratios = np.array([0.30, 0.60, 0.10])

    # --- Abordagem 1: Sequencial (Independente) ---
    # Compra (46, 4, 0) -> Venda (0, -27, -23)
    buy_seq = np.array([46, 4, 0])
    sell_seq = np.array([0, -27, -23])

    final_pos_seq = initial_pos + buy_seq + sell_seq
    final_ratios_seq = final_pos_seq / final_pos_seq.sum()

    # --- Abordagem 2: Conjunta (Simultânea) ---
    # Compra (50, 0, 0) -> Venda (0, -25, -25)
    buy_joint = np.array([50, 0, 0])
    sell_joint = np.array([0, -25, -25])
```

```
    final_pos_joint = initial_pos + buy_joint + sell_joint
    final_ratios_joint = final_pos_joint / final_pos_joint.sum()

    # --- 2. Plotagem ---
    plt.style.use('bmh')
    fig, ax = plt.subplots(figsize=(14, 8))

    indices = np.arange(len(funds))
    width = 0.2

    # Dados para as barras
    data_series = [
        (initial_pos, '1. Posição Inicial', 'gray', 0.5, 'normal'),
        (final_pos_seq, '2. Abordagem Sequencial', 'tab:orange', 1.0,
'normal'),
        (final_pos_joint, '3. Abordagem Conjunta', 'tab:blue', 1.0,
'bold'),
        (ideal_ratios * total_assets, '4. Meta Ideal', 'green', 0.3,
'dashed')
    ]

    rects_list = []

    for i, (data, label, color, alpha, style) in
enumerate(data_series):
        edge_color = 'green' if style == 'dashed' else None
        line_style = '--' if style == 'dashed' else '-'

        rects = ax.bar(indices + (i - 1.5) * width,
                       data / total_assets * 100,
                       width,
                       label=label,
                       color=color,
                       alpha=alpha,
                       edgecolor=edge_color,
                       linestyle=line_style)

        rects_list.append((rects, data))

    # --- 3. Customização Visual ---
    ax.set_ylabel('Alocação (%)')
    ax.set_title('Comparativo: Impacto em Unidades e Porcentagem\
n(Cenário: Compra +50 / Venda -50)', fontsize=14)
    ax.set_xticks(indices)
    ax.set_xticklabels(funds, fontsize=12)
    ax.legend(loc='upper right')
    ax.set_ylim(0, 80)

    # Função para Labels com Unidades
```

```
    def autolabel(rects, values_units, is_bold=False):
        for rect, unit_val in zip(rects, values_units):
            height = rect.get_height()
            weight = 'bold' if is_bold else 'normal'
            label_text = f'{height:.1f}%\n({int(unit_val)})'

            ax.annotate(label_text,
                        xy=(rect.get_x() + rect.get_width() / 2,
height),
                        xytext=(0, 5),
                        textcoords="offset points",
                        ha='center', va='bottom', fontsize=9,
fontweight=weight)

    # Aplicando rótulos
    autolabel(rects_list[1][0], rects_list[1][1]) # Sequencial
    autolabel(rects_list[2][0], rects_list[2][1], is_bold=True) #
Conjunta
    autolabel(rects_list[3][0], rects_list[3][1]) # Ideal

    # --- 4. Destaque (Storytelling) ---
    # Calcular a diferença em unidades no Fundo 1
    # Conjunta (300) - Sequencial (296) = 4 unidades
    diff_units = rects_list[2][1][0] - rects_list[1][1][0]

    # Correção do erro: removi o 'shrink' que estava conflitando
    ax.annotate(f'Diferença de {int(diff_units)} unidades\nperdididas
na ineficiência',
                xy=(indices[0], 31), xytext=(indices[0]+0.35, 40),
                arrowprops=dict(facecolor='black', arrowstyle="->",
connectionstyle="angle3,angleA=0,angleB=-90"),
                bbox=dict(boxstyle="round,pad=0.3", fc="white",
ec="black", lw=1),
                fontsize=10)

    plt.tight_layout()
    plt.show()

plot_comparacao_estrategias_com_unidades_v2()
```




```python
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np

def plot_min_piece_schematic():
    # Configuração
    fig, ax = plt.subplots(figsize=(16, 4))
    ax.set_ylim(-1, 2)
    ax.set_xlim(-16, 16) # Coordenadas visuais (fictícias), não os
valores reais

    # Limpar eixos
    ax.axis('off')

    # ==========================================
    # 1. Definição dos Dados (Mapeamento Valor Real -> Posição Visual)
    # ==========================================
    # Vamos criar uma "posição visual" fictícia para cada número.
    # O Zero fica no 0.
    # O 100 fica na posição 5, o 101 na 6, etc (espaçamento de 1).
    # O -100 fica na posição -5, o -101 na -6, etc.

    # Lado Direito (Positivos)
    right_values = np.arange(100, 111, 1) # 100 a 110
    right_pos = np.arange(5, 16, 1)       # Posições visuais 5 a 15

    # Lado Esquerdo (Negativos)
    left_values = np.arange(-110, -99, 1)[::-1] # -100 a -110
```

```
(invertido para plotar da direita pra esq)
    left_pos = np.arange(-15, -4, 1)[::-1]      # Posições visuais -15
a -5

    # ==========================================
    # 2. Desenhar Zonas
    # ==========================================

    # Linha do Eixo X (Cinza claro)
    ax.axhline(0, color='lightgray', linewidth=1, zorder=1)

    # Zona Proibida (Retângulo Vermelho entre -5 e 5, visualmente)
    # Note que na vida real seria -100 a 100, mas visualmente
encurtamos isso
    rect = patches.Rectangle((-4.5, -0.6), 9, 1.2, color='mistyrose',
alpha=0.5, zorder=0)
    ax.add_patch(rect)

    ax.text(0, 0.8, "Zona Proibida\n(Zero é a única exceção)",
ha='center', color='tab:red', fontsize=10, fontweight='bold')
    ax.text(2.5, -0.4, "Salto do\nMin Piece", ha='center',
color='tab:red', fontsize=9, style='italic')
    ax.text(-2.5, -0.4, "Salto do\nMin Piece", ha='center',
color='tab:red', fontsize=9, style='italic')

    # ==========================================
    # 3. Plotar Pontos (Bolinhas)
    # ==========================================

    # Bolinhas Azuis (Valores Válidos)
    ax.scatter(right_pos, np.zeros_like(right_pos), s=150,
color='tab:blue', zorder=10)
    ax.scatter(left_pos, np.zeros_like(left_pos), s=150,
color='tab:blue', zorder=10)

    # Bolinha Verde (Zero)
    ax.scatter([0], [0], s=200, color='limegreen', edgecolors='black',
zorder=10, label='Zero (Permitido)')

    # ==========================================
    # 4. Rótulos dos Números
    # ==========================================

    # Loop para rotular cada ponto positivo
    for pos, val in zip(right_pos, right_values):
        weight = 'bold' if val == 100 else 'normal'
        size = 11 if val == 100 else 10
        ax.text(pos, -0.3, str(val), ha='center', va='top',
fontsize=size, fontweight=weight)
```

```
    # Loop para rotular cada ponto negativo
    for pos, val in zip(left_pos, left_values):
        weight = 'bold' if val == -100 else 'normal'
        size = 11 if val == -100 else 10
        ax.text(pos, -0.3, str(val), ha='center', va='top',
fontsize=size, fontweight=weight)

    # Rótulo do Zero
    ax.text(0, -0.3, "0", ha='center', va='top', fontsize=12,
fontweight='bold')

    # ==========================================
    # 5. Setas de Continuação (Infinito)
    # ==========================================

    # Seta Direita
    ax.annotate('', xy=(16.5, 0), xytext=(15.5, 0),
arrowprops=dict(arrowstyle="->", lw=2, color='tab:blue'))
    ax.text(17, 0, "...", ha='left', va='center', fontsize=14,
color='tab:blue', fontweight='bold')

    # Seta Esquerda
    ax.annotate('', xy=(-16.5, 0), xytext=(-15.5, 0),
arrowprops=dict(arrowstyle="->", lw=2, color='tab:blue'))
    ax.text(-17, 0, "...", ha='right', va='center', fontsize=14,
color='tab:blue', fontweight='bold')

    # ==========================================
    # 6. Destaque do Incremento (Bracket)
    # ==========================================
    # Desenhar um colchete visual entre 100 e 101 (posições 5 e 6)

    # Posições visuais
    x_start, x_end = 5, 6
    y_h = 0.25 # Altura do bracket

    # Linhas manuais para fazer o bracket {
    ax.plot([x_start, x_start, x_end, x_end], [0.15, y_h, y_h, 0.15],
color='gray', lw=1)
    ax.plot([(x_start+x_end)/2, (x_start+x_end)/2], [y_h, y_h+0.1],
color='gray', lw=1)

    ax.text((x_start+x_end)/2, y_h+0.2, "Passo Inteiro\n(1 un)",
ha='center', va='bottom', color='gray', fontsize=9)

    plt.title("Domínio Discreto (Ilustrativo): Lote Mínimo de 100",
fontsize=15, pad=20)
    plt.tight_layout()
    plt.show()
```

```
plot_min_piece_schematic()
```


```python
def optimize(
    position: npt.NDArray[np.int64],
    ideal_ratios: npt.NDArray[np.float64],
    positive_amount: int,
    negative_amount: int,
    min_piece: int,
    min_increment: int,
):
    model = Model()
    # model.setParam("limits/time", 60)
    # model.setParam("numerics/feastol", 1e-7)

    scaled_positive_amount = positive_amount // min_increment
    scaled_negative_amount = negative_amount // min_increment
    scaled_position = position // min_increment
    scaled_min_piece = min_piece // min_increment

    num_funds = len(scaled_position)
    new_total_amount = scaled_position.sum() + scaled_positive_amount
+ scaled_negative_amount

    trades_positive = model.addMatrixVar(
        num_funds,
        name="trade_amounts_positive",
        ub=scaled_positive_amount,
        lb=0,
        vtype="I",
    )

    trades_negative = model.addMatrixVar(
        num_funds,
        name="trade_amounts_negative",
        ub=0,
        lb=scaled_negative_amount,
        vtype="I",
    )
```

```
    final_position = model.addMatrixVar(
        num_funds,
        name="final_position",
        lb=None,
        vtype="I",
    )

    model.addCons(trades_positive.sum() == scaled_positive_amount)
    model.addCons(trades_negative.sum() == scaled_negative_amount)

    model.addCons(final_position.sum() == new_total_amount)

    model.addMatrixCons(final_position == scaled_position +
trades_positive + trades_negative)

    min_piece_valid = model.addMatrixVar(
        num_funds,
        name="min_piece",
        vtype="B",
    )

    is_positive_dir = model.addMatrixVar(
        num_funds,
        name="is_positive_dir",
        vtype="B",
    )

    # Essa seguinte parte serve para modelar restrições do tipo OU
usando restrições lineares
    # -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    big_m = np.maximum(
        abs(scaled_position + scaled_positive_amount),
        abs(scaled_position + scaled_negative_amount),
    ) + scaled_min_piece

    # Se min_piece_valid = 0, então final_position = 0
    model.addMatrixCons(is_positive_dir <= min_piece_valid)

    # Se min_piece_valid = 1, então |final_position| <= big_m que é
igual a -big_m <= final_position <= big_m
    model.addMatrixCons(final_position <= big_m * min_piece_valid)
    model.addMatrixCons(final_position >= -big_m * min_piece_valid)

    # Se min_piece_valid = 1 e is_positive_dir = 1, então
final_position >= scaled_min_piece
    model.addMatrixCons(final_position >= scaled_min_piece *
min_piece_valid - big_m * (1 - is_positive_dir))

    # Se min_piece_valid = 1 e is_positive_dir = 0, então
```

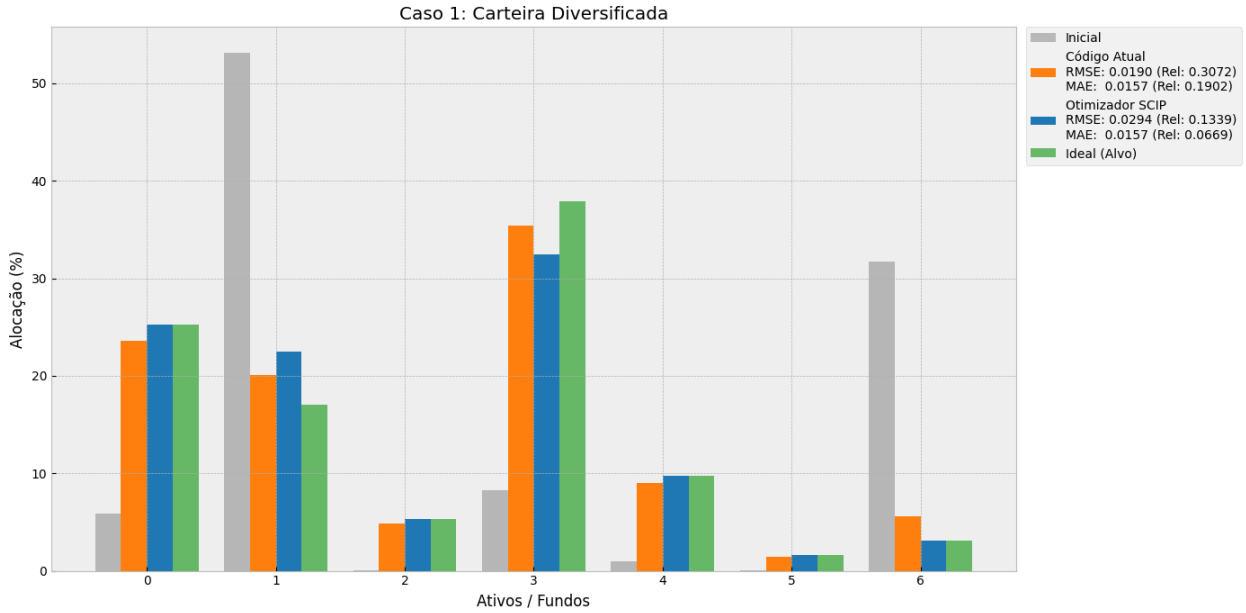
```
final_position <= -scaled_min_piece
    model.addMatrixCons(final_position <= -scaled_min_piece *
min_piece_valid + big_m * is_positive_dir + big_m * (1 -
min_piece_valid))

    # -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

    new_ratios = model.addMatrixVar(
        num_funds,
        name="new_ratios",
        lb=0,
        vtype="C",
    )

    model.addMatrixCons(new_total_amount * new_ratios ==
final_position)

    ratios_errors = model.addMatrixVar(
        num_funds,
        name="ratios_errors",
        lb=0,
        vtype="C",
    )

    z = model.addVar(name="z", vtype="C")

    # ratios_errors = (new_ratios - ideal_ratios) / ideal_ratios

    # L1
    model.addMatrixCons(ideal_ratios * ratios_errors >= new_ratios -
ideal_ratios)
    model.addMatrixCons(ideal_ratios * ratios_errors >= -(new_ratios -
ideal_ratios))

    # L2
    # model.addMatrixCons(ideal_ratios ** 2 * ratios_errors ==
(new_ratios - ideal_ratios) ** 2)

    expr = quicksum(
        ratios_errors[fund]
        for fund in range(num_funds)
    )

    scale_factor = 10000

    model.addCons(z >= expr * scale_factor)

    model.setObjective(z, "minimize")
```

```
    model.optimize()

    status = model.getStatus()
    print(f"Status: {status}")

    if status == "infeasible":
        print("\nPROBLEMA INFEASÍVEL!")
        print("\nAnalisando conflito...")
        model.writeProblem("debug_model.lp")
        print("Modelo salvo em: debug_model.lp")
        return None, None


    return model.getVal(trades_positive) * min_increment,
model.getVal(trades_negative) * min_increment

def calcular_erros(current_ratios, ideal_ratios):
    diff = current_ratios - ideal_ratios
    diff_relativo = diff / ideal_ratios

    # Mean Absolute Error
    mae = np.abs(diff).mean()
    # Mean Absolute Relative Error
    mae_rel = np.abs(diff_relativo).mean()

    # Root Mean Square Error
    rmse = np.sqrt((diff ** 2).mean())
    # Root Mean Square Relative Error
    rmse_rel = np.sqrt((diff_relativo ** 2).mean())

    return mae, mae_rel, rmse, rmse_rel

def analisar_cenario(title, position, ideal_ratios, pos_amt, neg_amt,
min_p, min_inc,
                     trades_pos_ref, trades_neg_ref, label_ref="Código
Atual"):

    # 1. Calcular cenário de Referência (Hardcoded/Código Atual)
    pos_ref = position + trades_pos_ref + trades_neg_ref
    ratios_ref = pos_ref / pos_ref.sum()

    # Desempacotando os 4 valores
    mae_ref, mae_rel_ref, rmse_ref, rmse_rel_ref =
calcular_erros(ratios_ref, ideal_ratios)

    # 2. Rodar o Otimizador
    print(f"--- Otimizando: {title} ---")
    opt_pos, opt_neg = optimize(position, ideal_ratios, pos_amt,
neg_amt, min_p, min_inc)
```

```
    if opt_pos is None:
        print("Otimização falhou.")
        return

    pos_opt = position + opt_pos + opt_neg
    ratios_opt = pos_opt / pos_opt.sum()

    # Desempacotando os 4 valores
    mae_opt, mae_rel_opt, rmse_opt, rmse_rel_opt =
calcular_erros(ratios_opt, ideal_ratios)

    # 3. Plotagem
    indices = np.arange(len(position))
    width = 0.2

    plt.figure(figsize=(14, 7))

    # Posição Inicial
    initial_ratios = position / position.sum()
    plt.bar(indices - 1.5*width, initial_ratios * 100, width,
label='Inicial', color='gray', alpha=0.5)

    # Referência (Código Atual) - Formatando legenda com quebra de
linha
    label_ref_full = (f"{label_ref}\n"
                      f"RMSE: {rmse_ref:.4f} (Rel:
{rmse_rel_ref:.4f})\n"
                      f"MAE:  {mae_ref:.4f} (Rel: {mae_rel_ref:.4f})")

    plt.bar(indices - 0.5*width, ratios_ref * 100, width,
label=label_ref_full, color='tab:orange')

    # Otimizador - Formatando legenda com quebra de linha
    label_opt_full = (f"Otimizador SCIP\n"
                      f"RMSE: {rmse_opt:.4f} (Rel:
{rmse_rel_opt:.4f})\n"
                      f"MAE:  {mae_opt:.4f} (Rel: {mae_rel_opt:.4f})")

    plt.bar(indices + 0.5*width, ratios_opt * 100, width,
label=label_opt_full, color='tab:blue')

    # Ideal
    plt.bar(indices + 1.5*width, ideal_ratios * 100, width,
label='Ideal (Alvo)', color='tab:green', alpha=0.7)

    plt.ylabel('Alocação (%)')
    plt.xlabel('Ativos / Fundos')
    plt.title(title)
    plt.xticks(indices)
```

```
    # Legenda posicionada fora
    plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left',
borderaxespad=0.)

    plt.tight_layout()
    plt.show()

    # Print textual para conferência rápida
    print(f"Trades Otimizador (+): {opt_pos}")
    print(f"Trades Otimizador (-): {opt_neg}")
    print("-" * 50)

# ==========================================
# CASO 1: 7 Ativos
# ==========================================
pos1 = np.array([9902, 89787, 20, 14008, 1689, 38, 53556],
dtype=np.int64)
ideal1 = np.array([0.2523677372486256, 0.17010906099149858,
0.053109950549653805,
                   0.379392770883289, 0.09772944240523304,
0.01592742600174438,
                   0.031363611919955785], dtype=np.float64)
# Valores de referência (código atual)
ref_pos1 = np.array([29963, 0, 8194, 45848, 13567, 2428, 0])
ref_neg1 = np.array([0, -55848, 0, 0, 0, 0, -44152])

analisar_cenario(
    title="Caso 1: Carteira Diversificada",
    position=pos1, ideal_ratios=ideal1,
    pos_amt=100000, neg_amt=-100000, min_p=1, min_inc=1,
    trades_pos_ref=ref_pos1, trades_neg_ref=ref_neg1
)
```

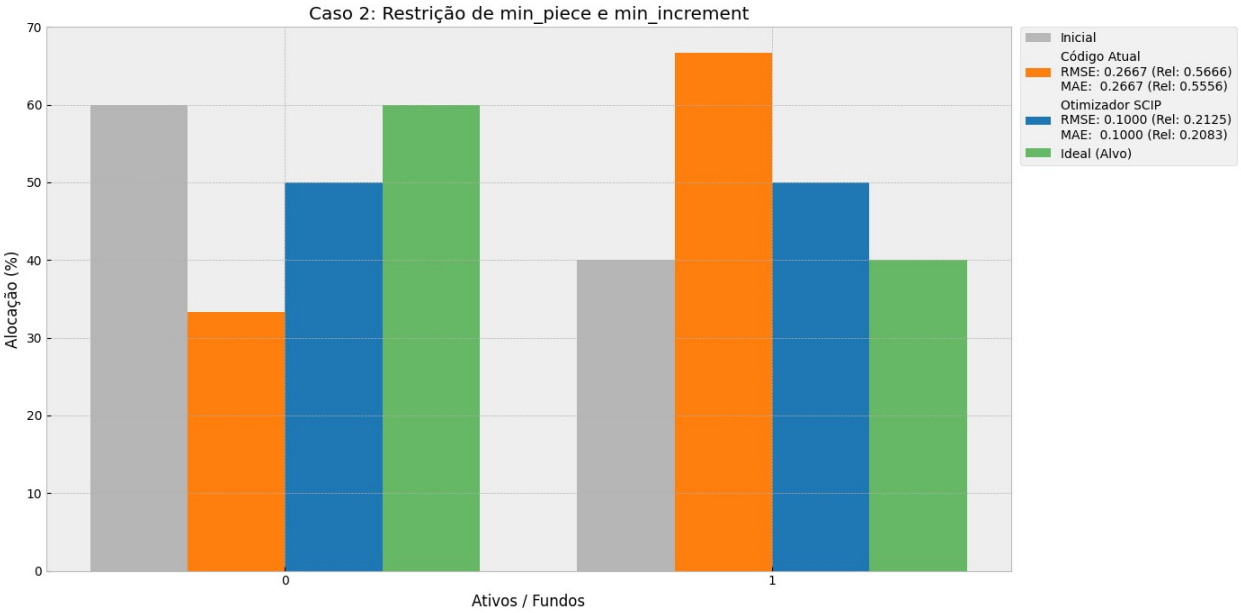
```
--- Otimizando: Caso 1: Carteira Diversificada ---
Status: optimal
```




```
Trades Otimizador (+): [32748.     0.  8956. 40815. 14827.  2654.
0.]
Trades Otimizador (-): [     0. -51745.      0.      0.      0.
0. -48255.]
--------------------------------------------------
```

```python
# ==========================================
# CASO 2: 2 Ativos (Min Increment 50)
# ==========================================
pos2 = np.array([150, 100], dtype=np.int64)
ideal2 = np.array([0.6, 0.4], dtype=np.float64)
# Valores de referência (código atual)
ref_pos2 = np.array([0, 0])
ref_neg2 = np.array([-100, 0])

analisar_cenario(
    title="Caso 2: Restrição de min_piece e min_increment",
    position=pos2, ideal_ratios=ideal2,
    pos_amt=0, neg_amt=-100, min_p=75, min_inc=25,
    trades_pos_ref=ref_pos2, trades_neg_ref=ref_neg2
)
```

```
--- Otimizando: Caso 2: Restrição de min_piece e min_increment ---
Status: optimal
```




```
Trades Otimizador (+): [-0. -0.]
Trades Otimizador (-): [-75. -25.]
--------------------------------------------------
```