Příloha 3

Kódové provedení Abbottovo-Firestonovo křivky

Hlavní spouštěcí funkcí je bearing_ratio_curve.

```
def bearing_ratio(data, real_step=0.01):
    ubound = int(real_step**-1)

    y_axis = np.linspace(max(data), min(data), num=ubound*2)

    first_half_data = data.copy()
    first_half_data[data<0] = 0

    second_half_data = -data.copy()
    second_half_data[data>=0] = 0

    first_half_volume = np.trapz(first_half_data)
    second_half_volume = np.trapz(second_half_data)
    max_volume = first_half_volume + second_half_volume

    max_value = max(data)
    min_value = -min(data)

    bearing_ratio = np.zeros(2*ubound)

    for step in range(1, ubound+1):
        time_eater = data.copy()
        threshold = (ubound - step)/ubound*max_value
        time_eater[time_eater<threshold] = threshold
        time_eater -= threshold
        volume = np.trapz(time_eater)
        bearing_ratio[step-1] = round(volume/max_volume*100, 1)
        last_vol = volume



    for s in range(1, ubound+1):
        step = ubound - s
        time_eater = -1*data.copy()
        threshold = (ubound - step)/ubound*min_value
        time_eater[time_eater<threshold] = threshold
        time_eater[time_eater<0] = 0
        time_eater -= threshold
        volume = last_vol + second_half_volume - np.trapz(time_eater,
                                                    dx=step/ubound)
        bearing_ratio[-(step+1)+ubound*2] = round(volume/max_volume*100, 1)

    return bearing_ratio, y_axis


def find_smallest_gradient(data, window_size=0.05):
    np.seterr(invalid='ignore')
    window = int(len(data)*window_size)
    num_of_steps = int(len(data)/window)

    output_data = {'x-val' : np.zeros(num_of_steps),
                   'y-val' : np.zeros(num_of_steps),
                   'Correlation' : np.zeros(num_of_steps),
                   'Corr_diff' : np.zeros(num_of_steps),
                   'Angle' : np.zeros(num_of_steps),
                   'Group': np.zeros(num_of_steps)}

    for step in range(num_of_steps):
        lower_X = window*step
        higher_X = window*step+window if num_of_steps-1 != step else
                                                             len(data)-1
        middle_X = int((higher_X - lower_X)/2) + lower_X

        output_data['x-val'][step] = middle_X
        output_data['y-val'][step] = data[middle_X]
        x1=lower_X
        x2=higher_X
        y1=data[lower_X]
        y2=data[higher_X]

        output_data['Angle'][step] = np.arctan((y2-y1)/(x2-x1))/np.pi*180
        output_data['Corr_diff'][step] = round(output_data['Angle'][step] -
                                           output_data['Angle'][step-1], 4)

    return pd.DataFrame(output_data)


def smallest_gradient_line(df, threshold=5, step=0.01, crop=0.35):
    # Concition n0, crop 10% of edge values
    cropper = int(len(df)*crop)
    df0 = df[cropper:-cropper].copy()

    # Condition n1, difference lower than threshold
    df1 = df0[abs(df0['Corr_diff']) < threshold].copy()

    # Condition n2, continuity & ignore low values
    prev_idx = df1.index[0]
    group = 0



    for idx in df1.index:
        if (idx == 0) | (idx == 1):
            df1.loc[idx, 'Group'] = group
            group += 1
        elif idx == prev_idx+1:
            df1.loc[idx, 'Group'] = group
        else:
            group += 1
            df1.loc[idx, 'Group'] = group
        prev_idx = idx

    # Condition n3, dataset with largest continuity
    largest_group = df1['Group'].value_counts().index.values[0]
    df2 = df1[df1['Group']==largest_group]

    # Condition n4 only third of values is enough to calculate angle
    third = int(len(df2)/3)
    df3 = df2[third:-third].copy()

    # Create line from first and last point y = m.x + b
    calculated_angle = np.mean(df3['Angle'])
    m = np.tan(calculated_angle/180*np.pi)
    b = np.mean(df3['y-val'])-m*np.mean(df3['x-val'])

    x = np.arange(1, int(step**-1)*2+1, 1)
    line = x*m + b
    line[line<0] = None
    line[line>100] = None

    return line


def bearing_ratio_curve(real_data, window_size=0.02, threshold=5):
    """
    Bearing ratio curve generates main roughness coeficients from
    Abbott-Firestone curve. Such as Rpk, Rk, Rvk, Mr1, Mr2

    Parameters
    ------------
    real_data : pandas.DataFrame
        Signal from which the coefficients will be calculated
    window_size : float
        The gradient line is calculated by floating window. Larger window
        will
        speed up the calculation but lower the quality of the outputs.
        Recommended: 0.02
    threshold : int
        Condition of smallest gradient difference is determined by a fixed
        threshold.
        Recommended: 5

    Returns
    ----------
    DataFrame with calculated coefficients Rpk, Rk, Rvk, Mr1, Mr2

    """
    ac_data = real_data.copy()
    bearing_rat, y_data = bearing_ratio(ac_data, real_step=0.01)
    _, filtered = tools.return_roughness(bearing_rat.reshape(-1, 1),
                                         filter_order=1, crit_freq=.05)

    gradient_data = find_smallest_gradient(filtered,
                                           window_size=window_size)

    line = smallest_gradient_line(gradient_data, threshold=threshold)

    p1=y_data[line==line[~np.isnan(line)][0]]
    p2=y_data[line==line[~np.isnan(line)][-1]]

    Rpk = y_data[0]-p1 #Reduced peak height
    Rk = p1-p2 #Core roughness depth
    Rvk = p2 - y_data[-1] #Reduced valley depth
    Mr1 = bearing_rat[y_data==p1] #Peak material portion
    Mr2 = bearing_rat[y_data==p2] #Valley material portion

    return pd.DataFrame({'name': [Rpk[0], Rk[0], Rvk[0], Mr1[0], Mr2[0]]},
                        index=['Rpk', 'Rk', 'Rvk', 'Mr1', 'Mr2'])
```