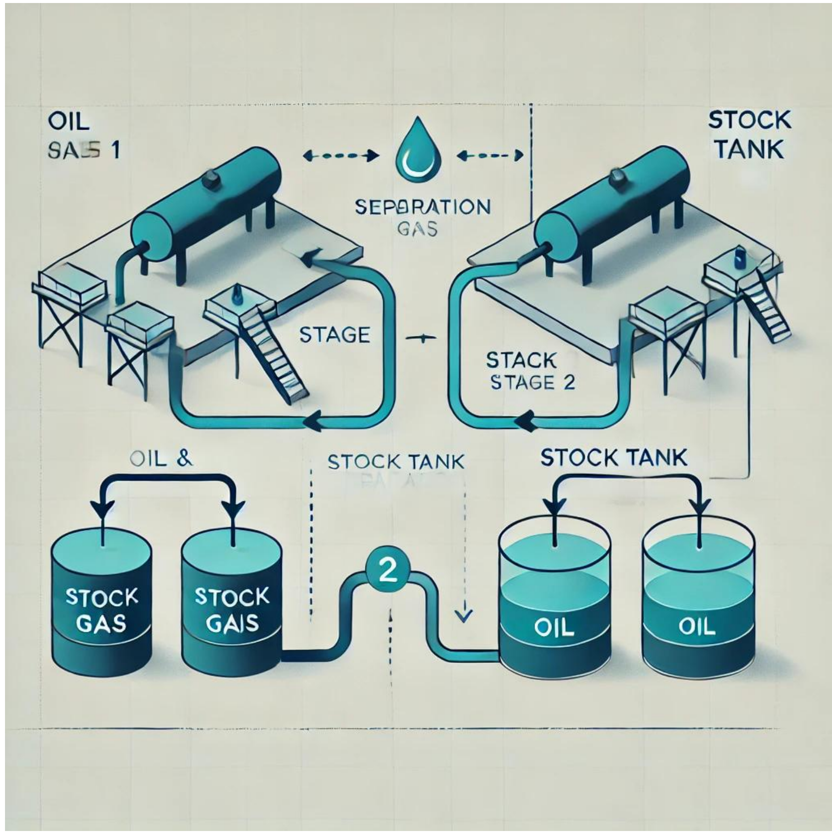
Automating DLE Correction for Reservoir Engineering: A Python-Based Approach to Enhance PVT Data Accuracy



Solomon U. Okoro
Terry Idahor

www.theuvere.com/projects

March 2025

Introduction to DLE Correction

Definition:

- DLE (Differential Liberation Experiment) measures oil properties (Bo, Rs) under reservoir conditions.
- Correction adjusts DLE data to match field separator conditions.

Objective:

- Align lab data with actual production conditions for accurate reservoir analysis.

Importance:

- Critical for material balance, reservoir simulation, and production forecasting.

Introduction to DLE Correction

Why Perform DLE Correction?

Consistency:

- Lab DLE data (reservoir conditions) differs from field separator conditions.
- Correction ensures data consistency for field applications.

Accuracy:

- Improves reliability of Bo (oil formation volume factor) and Rs (solution gas-oil ratio).
- Enhances reserve estimation and recovery predictions.

Practical Impact:

- Optimizes production strategies.
- Reduces errors in field development planning.

Overview of the DLE Correction Script

Purpose:

- Automates DLE correction using CCE and Separator Test data.

Key Components:

- Global variables for data validation.
- Data import and parsing functions.
- Correction functions for calculations.
- Main function for workflow and visualization.

Libraries Used:

- NumPy, Pandas, Plotly express for data processing and plotting, .

Defining Expected Columns

- Variable: EXPECTED_COLS
- Purpose:
  - Specifies the number of columns for each dataset.
- Details:
  - CCE Experiment: 2 columns (Pressure (psia), Relative Volume (V/Vb)).
  - DLE Experiment: 3 columns (Pressure (psia), Bo (RB/STB), Rs (scf/stb)).
  - Separator Test: 4 columns (Separator Pressure (psia), Separator Temperature (degF), GOR (Rsfb) (scf/stb), Bofb (RB/STB))
- Benefit:
  - Validates input data structure to prevent errors.

```python
# Global expected columns and headers.
EXPECTED_COLS = {
    "CCE Experiment": 2,
    "DLE Experiment": 3,
    "Separator Test": 4
}
```

Defining Expected Headers

- Variable: EXPECTED_HEADERS

```
EXPECTED_HEADERS = {
    "CCE Experiment": ["Pressure (psia)", "Relative Volume (V/Vb)"],
    "DLE Experiment": ["Pressure (psia)", "Bo (RB/STB)", "Rs (scf/stb)"],
    "Separator Test": ["Separator Pressure (psia)", "Separator Temperature (degF)",
                       "GOR (Rsfb) (scf/stb)", "Bofb (RB/STB)"]
}
```

- Purpose:
  - Defines required column headers for each dataset.
- Details:
  - Ensures headers match expected formats (e.g., Pressure (psia)).
  - Overrides headers for CCE and DLE if column count matches.
  - Example:

    DLE: ["Pressure (psia)", "Bo (RB/STB)", "Rs (scf/stb)"].
- Benefit:
  - Standardizes data for consistent processing.

Data Import – Helper Functions

Functions:

- reassemble_rows_if_single_line: Converts single-line input into multi-line format based on expected columns.
- read_data_until_blank_line: Reads multi-line input until a blank line.
- insert_newline_if_missing: Placeholder for newline handling (used in parsing).

Purpose:

- Handles various input formats (single-line or multi-line).
- Ensures robust data parsing for user convenience.

Data Import – Parsing Functions

Main Function:

parse_data_from_string

Process:

- Detects separators (tabs, commas, spaces) in the input.
- Parses data into a Pandas DataFrame.
- Overrides headers for CCE and DLE if column count matches expected.

Error Handling:

- Raises errors for invalid formats or missing data.

Example Output:

- Parsed CCE DataFrame with columns: Pressure (psia), Relative Volume (V/Vb).

Data Import – User Prompting

Function prompt_for_data

Purpose:

- Guides the user on data input format and units.

Details:

- Displays expected units (e.g.,

  Pressure in psia, Bo in RB/STB).

- Provides an example dataset for clarity.
- Special handling for Separator Test (no header row required).

Benefit:

- Improves user experience with clear instructions.

Correction Functions - Bubble Point Validation

Function

validate_bubble_point

Purpose:

- Ensures the bubble-point pressure (pb) is a valid number.

Process:

- Prompts user for input.
- Validates input as a float or integer.
- Loops until a valid value is provided.

```
def validate_bubble_point():
    while True:
        try:
            pb = float(input("Enter the bubble-point pressure (psia, integer or float): "))
            return pb
        except ValueError:
            print("Error: Bubble-point pressure must be a number. Try again.")
```

Example

For Well AB: pb = 1662 psia.

Correction Functions – DLE Correction Logic

Function

correct_dle_to_separator

Inputs:

- DLE data: pressures, bo_dle, rs_dle.
- CCE data: vrel (relative volume).
- Separator data: bofb, rsfb.
- Bubble-point pressure: pb.

Key Steps:

- Finds bubble-point index in DLE data.
- Computes Sod = Bo/Bodb (Bo at bubble-point).

Output:

- Corrected Bo and Rs values.

```python
def correct_dle_to_separator(pressures, bo_dle, rs_dle, vrel, pb, rsfb, bofb):
    """
    Corrects DLE data to separator conditions.
    Uses the bubble-point pressure and the optimum separator condition (Bofb, Rsfb)
    to compute corrected DLE Bo and Rs.
    """
    pb_idx = np.where(np.isclose(pressures, pb, atol=1e-2))[0]
    if len(pb_idx) == 0:
        raise ValueError(f"Bubble-point pressure {pb} not found in DLE pressure data: {pressures}")
    pb_idx = pb_idx[0]
    bodb = bo_dle[pb_idx]     # Bo at bubble-point
    rsds = rs_dle[pb_idx]     # Rs at bubble-point
    sod = bo_dle / bodb
    bo_corrected = np.zeros_like(bo_dle)
    for i, p in enumerate(pressures):
        if p >= pb:
            bo_corrected[i] = vrel[i] * bofb
        else:
            bo_corrected[i] = bofb * sod[i]
    rs_corrected = np.zeros_like(rs_dle)
    for i, p in enumerate(pressures):
        if p >= pb:
            rs_corrected[i] = rsfb
        else:
            rs_corrected[i] = rsfb - (rsds - rs_dle[i]) * (bofb / bodb)
    return bo_corrected, rs_corrected
```

Correction Calculations – Bo Correction

Logic:

- Above bubble-point (p >= pb):

  Bo_corrected = Vrel * Bofb.

  Uses CCE relative volume to scale Bofb.

- Below bubble-point (p < pb):

  Bo_corrected = Bofb * Sod.

  Sod = Bo/Bodb (ratio of Bo to Bo at bubble-point).

```python
def correct_dle_to_separator(pressures, bo_dle, rs_dle, vrel, pb, rsfb, bofb):
    """
    Corrects DLE data to separator conditions.
    Uses the bubble-point pressure and the optimum separator condition (Bofb, Rsfb)
    to compute corrected DLE Bo and Rs.
    """
    pb_idx = np.where(np.isclose(pressures, pb, atol=1e-2))[0]
    if len(pb_idx) == 0:
        raise ValueError(f"Bubble-point pressure {pb} not found in DLE pressure data: {pressures}")
    pb_idx = pb_idx[0]
    bodb = bo_dle[pb_idx]    # Bo at bubble-point
    rsds = rs_dle[pb_idx]    # Rs at bubble-point
    sod = bo_dle / bodb
    bo_corrected = np.zeros_like(bo_dle)
    for i, p in enumerate(pressures):
        if p >= pb:
            bo_corrected[i] = vrel[i] * bofb
        else:
            bo_corrected[i] = bofb * sod[i]
    rs_corrected = np.zeros_like(rs_dle)
    for i, p in enumerate(pressures):
        if p >= pb:
            rs_corrected[i] = rsfb
        else:
            rs_corrected[i] = rsfb - (rsds - rs_dle[i]) * (bofb / bodb)
    return bo_corrected, rs_corrected
```

Purpose:

- Adjusts Bo to separator conditions.

Example:

At p = 5015 psia, Vrel = 0.9632, Bofb = 1.334 → Bo_corrected = 0.9632 * 1.334.

Correction Calculations – Rs Correction

Logic:

- Above bubble-point (p >= pb):

  Rs_corrected = Rsfb

  Rs remains constant above Pb.

```
rs_corrected = np.zeros_like(rs_dle)
for i, p in enumerate(pressures):
    if p >= pb:
        rs_corrected[i] = rsfb
    else:
        rs_corrected[i] = rsfb - (rsds - rs_dle[i]) * (bofb / bodb)
return bo_corrected, rs_corrected
```

- Below bubble-point (p < pb):

  Rs_corrected = Rsfb - (Rsds - Rsd) * (Bofb/Bodb)

  Rsds: Rs at bubble-point; Rsd: Rs at current pressure.

Purpose:

- Adjusts Rs to separator conditions.

Example:

At p = 1515 psia, Rsfb = 494, Rsds = 563, Rsd = 522.2 → Adjusted Rs

Main Function – Workflow

Steps:

1. Imports CCE, DLE, and Separator Test data.
2. Validates bubble-point pressure.
3. Extracts optimum separator conditions (Bofb, Rsfb).
4. Merges CCE Vrel with DLE data.
5. Performs DLE correction.
6. Visualizes and exports results into excel file.

Key Feature:

Robust error handling for missing or invalid data.

Main Function – Visualization with Plotly Express

Plots:

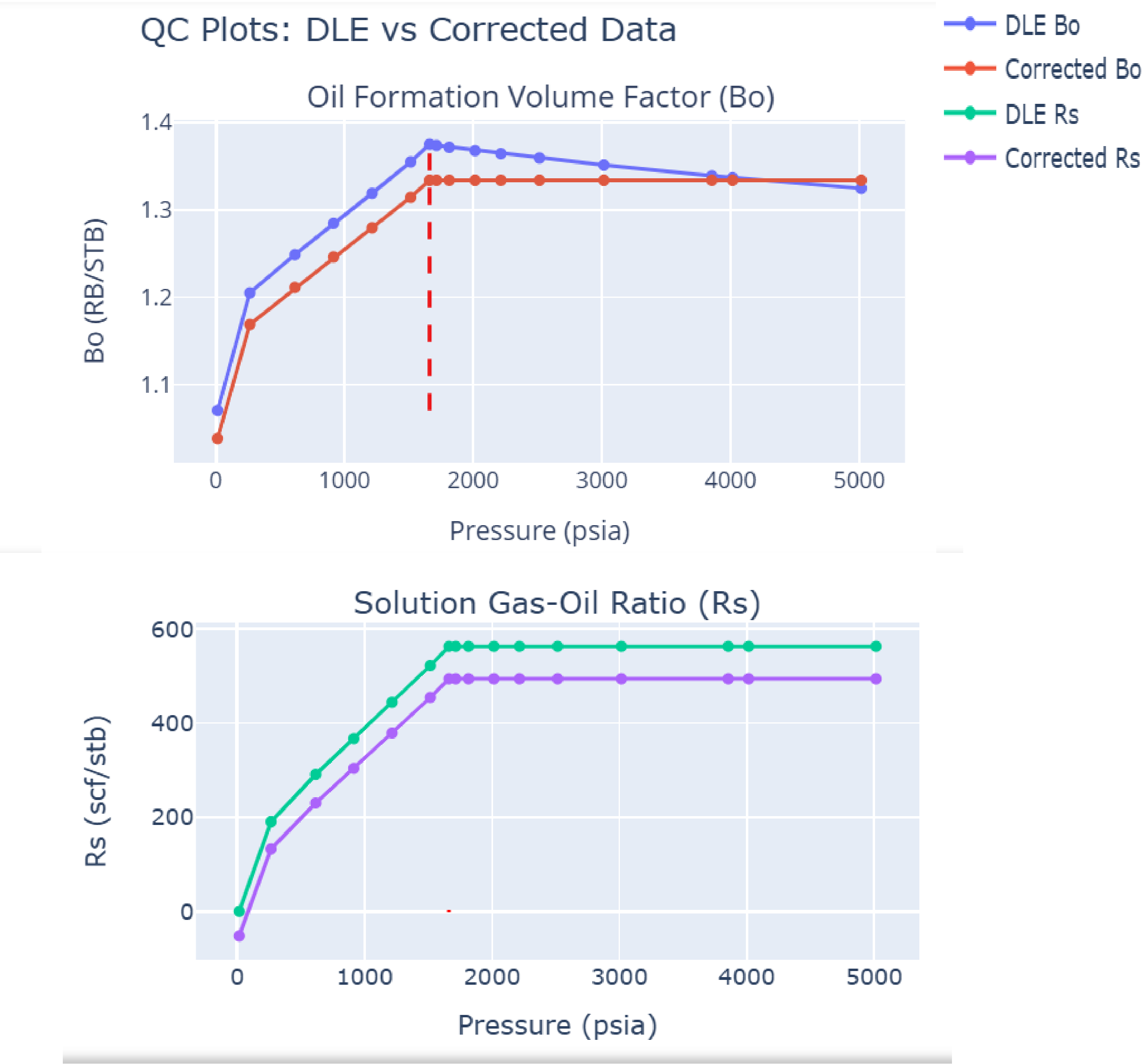
- Two subplots: Bo vs. Pressure, Rs vs. Pressure.
- DLE vs. corrected values with a vertical line at pb.

Purpose:

- Quality control (QC) to verify correction accuracy.

Benefit:

- Visual confirmation of correction impact.



Efficiency and Practical Benefits of DLE Correction

Automation:

- Reduces manual correction time from hours to minutes.

Accuracy:

- Aligns DLE data with field conditions, improving simulation reliability.

Practical Impact:

- Better reserve estimation and production optimization.

Example (Well AB):

Corrected Bo at 5015 psia: Reduced from 1.3247 to 1.2849 RB/STB.

Ensures accurate material balance calculations.

Scalability:

- Handles large datasets efficiently with Pandas.

References

- Ahmed, T. (2018). Reservoir engineering handbook (5th ed.). Gulf Professional Publishing.
- Dake, L. P. (2001). Fundamentals of reservoir engineering (2nd ed.). Elsevier.
- McCain, W. D. (1991). Reservoir-fluid property correlations—State of the art. SPE Reservoir Engineering, 6(2), 266–272. https://doi.org/10.2118/18571-PA
- Whitson, C. H., & Brulé, M. R. (2000). Phase behavior. SPE Monograph Series, Volume 20.

Questions?