extraction

February 11, 2025

[1]:
```
pip install pandoc
```

```
Collecting pandoc
  Downloading pandoc-2.4.tar.gz (34 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting plumbum (from pandoc)
  Downloading plumbum-1.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: ply in c:\users\dell\anaconda3\lib\site-packages
(from pandoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\dell\anaconda3\lib\site-
packages (from plumbum->pandoc) (305.1)
Downloading plumbum-1.9.0-py3-none-any.whl (127 kB)
Building wheels for collected packages: pandoc
  Building wheel for pandoc (setup.py): started
  Building wheel for pandoc (setup.py): finished with status 'done'
  Created wheel for pandoc: filename=pandoc-2.4-py3-none-any.whl size=34821
sha256=af42685c2b70128981bde2c101bf0ba822b776e6181a648904983454ea8ad130
  Stored in directory: c:\users\dell\appdata\local\pip\cache\wheels\9c\2f\9f\b1a
ac8c3e74b4ee327dc8c6eac5128996f9eadf586e2c0ba67
Successfully built pandoc
Installing collected packages: plumbum, pandoc
Successfully installed pandoc-2.4 plumbum-1.9.0
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-
packages (1.26.4)
Note: you may need to restart the kernel to use updated packages.
```

[6]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\dell\anaconda3\lib\site-
packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in
c:\users\dell\anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\dell\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
```

```
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-
packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\dell\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
pip install numpy pandas scipy
```

```
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-
packages (1.26.4)
Requirement already satisfied: pandas in c:\users\dell\anaconda3\lib\site-
packages (2.2.2)
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-
packages (1.13.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\dell\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-
packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\dell\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
import numpy as np
import pandas as pd
import tkinter as tk
from tkinter import ttk, messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

# --- Simulation Functions (same as before) ---
def extraction_efficiency(roller_gap, roller_surface_area, permeability,␣
 ↪juice_viscosity, juice_flow_rate):
    pressure_drop = (juice_viscosity * juice_flow_rate) / (permeability *␣
 ↪roller_surface_area)
    normalized_gap = roller_gap / 0.1
    normalized_permeability = permeability / 1e-10
    normalized_viscosity = juice_viscosity / 0.001
    efficiency = 1 - np.exp(- (1 / normalized_gap) * normalized_permeability /␣
 ↪normalized_viscosity)
    return np.clip(efficiency, 0, 1)

def darcy_number(permeability, roller_gap):
    return permeability / roller_gap**2
```

```
def juice_flow_rate(permeability, pressure_drop, roller_surface_area,␣
 ↪juice_viscosity, roller_gap):
    return (permeability * roller_surface_area * pressure_drop) /␣
 ↪(juice_viscosity * roller_gap)

def mass_balance(input_cane_mass, input_cane_pol, input_cane_brix,␣
 ↪extraction_efficiency_pol, extraction_efficiency_brix, imbibition_rate = 0,␣
 ↪imbibition_pol = 0, imbibition_brix = 0):
    extracted_pol = input_cane_mass * (input_cane_pol / 100) *␣
 ↪extraction_efficiency_pol
    extracted_brix = input_cane_mass * (input_cane_brix / 100) *␣
 ↪extraction_efficiency_brix
    juice_mass = extracted_brix
    juice_mass += imbibition_rate
    extracted_pol += imbibition_rate * (imbibition_pol / 100)
    extracted_brix += imbibition_rate * (imbibition_brix / 100)
    if juice_mass > 0:
        juice_pol = (extracted_pol / juice_mass) * 100
        juice_brix = (extracted_brix / juice_mass) * 100
    else:
        juice_pol = 0
        juice_brix = 0
    bagasse_mass = input_cane_mass - (juice_mass - imbibition_rate)
    remaining_pol = input_cane_mass * (input_cane_pol / 100) - extracted_pol
    remaining_brix = input_cane_mass * (input_cane_brix / 100) - extracted_brix
    if bagasse_mass > 0:
        bagasse_pol = (remaining_pol / bagasse_mass) * 100
        bagasse_brix = (remaining_brix / bagasse_mass) * 100
    else:
        bagasse_pol = 0
        bagasse_brix = 0
    return juice_mass, juice_pol, juice_brix, bagasse_mass, bagasse_pol,␣
 ↪bagasse_brix

def adjust_permeability_based_on_preparation(cane_preparation_index):
    base_permeability = 1e-10
    adjustment_factor = 1 + (cane_preparation_index - 0.5) * 0.5
    return base_permeability * adjustment_factor

def multi_roller_extraction(cane_mass, cane_pol, cane_brix, roller_configs,␣
 ↪imbibition_rates=None, imbibition_pol = 0, imbibition_brix = 0):
    if imbibition_rates is None:
        imbibition_rates = [0] * len(roller_configs)
    if len(imbibition_rates) != len(roller_configs):
```

```
        raise ValueError("Length of imbibition_rates must match the number of roller configurations.")
    stage_data = []
    current_cane_mass = cane_mass
    current_cane_pol = cane_pol
    current_cane_brix = cane_brix

    for i, config in enumerate(roller_configs):
        efficiency = extraction_efficiency(
            config['roller_gap'],
            config['roller_surface_area'],
            config['permeability'],
            config['juice_viscosity'],
            juice_flow_rate(config['permeability'], config['pressure_drop'], config['roller_surface_area'], config['juice_viscosity'], config['roller_gap'])
        )
        juice_mass, juice_pol, juice_brix, bagasse_mass, bagasse_pol, bagasse_brix = mass_balance(
            current_cane_mass,
            current_cane_pol,
            current_cane_brix,
            efficiency,
            efficiency,
            imbibition_rate=imbibition_rates[i],
            imbibition_pol= imbibition_pol,
            imbibition_brix = imbibition_brix
        )

        stage_data.append({
            'Stage': i + 1,
            'Juice Mass (kg)': juice_mass,
            'Juice Pol (%)': juice_pol,
            'Juice Brix (%)': juice_brix,
            'Bagasse Mass (kg)': bagasse_mass,
            'Bagasse Pol (%)': bagasse_pol,
            'Bagasse Brix (%)': bagasse_brix,
            'Extraction Efficiency': efficiency,
            'Darcy Number': darcy_number(config['permeability'], config['roller_gap'])
        })

        current_cane_mass = bagasse_mass
        current_cane_pol = bagasse_pol
        current_cane_brix = bagasse_brix

    return pd.DataFrame(stage_data)
```

```
# --- GUI Class ---
class SugarcaneApp:
    def __init__(self, master):
        self.master = master
        master.title("Sugarcane Extraction Simulator")

        # --- Input Variables (Tkinter Variables) ---
        self.cane_mass = tk.DoubleVar(value=1000.0)
        self.cane_pol = tk.DoubleVar(value=13.0)
        self.cane_brix = tk.DoubleVar(value=16.0)
        self.cane_preparation_index = tk.DoubleVar(value=0.7) # for␣
↪permeability adjustment
        self.imbibition_pol = tk.DoubleVar(value=0)
        self.imbibition_brix = tk.DoubleVar(value=0)
        self.num_stages = 3 # Set a default number of stages
        self.roller_gaps = [tk.DoubleVar(value=0.02) for _ in range(self.
↪num_stages)]
        self.roller_areas = [tk.DoubleVar(value=2.0) for _ in range(self.
↪num_stages)]
        self.juice_viscosities = [tk.DoubleVar(value=0.0015) for _ in␣
↪range(self.num_stages)]
        self.pressure_drops = [tk.DoubleVar(value=1e5) for _ in range(self.
↪num_stages)]
        self.imbibition_rates = [tk.DoubleVar(value=0.0) for _ in range(self.
↪num_stages)]

        # --- Input Frame ---
        self.input_frame = ttk.LabelFrame(master, text="Input Parameters",␣
↪padding=(10, 10))
        self.input_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

        # Labels and Entry fields for cane properties
        ttk.Label(self.input_frame, text="Cane Mass (kg):").grid(row=0,␣
↪column=0, sticky="w")
        ttk.Entry(self.input_frame, textvariable=self.cane_mass).grid(row=0,␣
↪column=1)
        ttk.Label(self.input_frame, text="Cane Pol (%):").grid(row=1, column=0,␣
↪sticky="w")
        ttk.Entry(self.input_frame, textvariable=self.cane_pol).grid(row=1,␣
↪column=1)
        ttk.Label(self.input_frame, text="Cane Brix (%):").grid(row=2,␣
↪column=0, sticky="w")
        ttk.Entry(self.input_frame, textvariable=self.cane_brix).grid(row=2,␣
↪column=1)
```

```
        ttk.Label(self.input_frame, text="Cane Prep. Index:").grid(row=3,␣
↪column=0, sticky="w")
        ttk.Entry(self.input_frame, textvariable=self.cane_preparation_index).
↪grid(row=3, column=1)
        ttk.Label(self.input_frame, text="Imbibition Pol (%):").grid(row=4,␣
↪column=0, sticky="w")
        ttk.Entry(self.input_frame, textvariable=self.imbibition_pol).
↪grid(row=4, column=1)
        ttk.Label(self.input_frame, text="Imbibition Brix (%):").grid(row=5,␣
↪column=0, sticky="w")
        ttk.Entry(self.input_frame, textvariable=self.imbibition_brix).
↪grid(row=5, column=1)


        # Stage Configuration Frame
        self.stage_frame = ttk.LabelFrame(master, text="Stage Configurations",␣
↪padding=(10, 10))
        self.stage_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

        self.stage_labels = [] # Store labels
        self.stage_entries = [] # Store entries

        # Create entries for each stage
        for i in range(self.num_stages):
            # Add Stage Header Label
            stage_label = ttk.Label(self.stage_frame, text=f"--- Stage {i+1}␣
↪---", font=('Helvetica', 10, 'bold'))
            stage_label.grid(row=i * 6, column=0, columnspan=2, sticky="w")  #␣
↪Adjusted row

            ttk.Label(self.stage_frame, text="Roller Gap (m):").grid(row=i * 6␣
↪+ 1, column=0, sticky="w") # Adjusted row
            ttk.Entry(self.stage_frame, textvariable=self.roller_gaps[i]).
↪grid(row=i * 6 + 1, column=1) # Adjusted row
            ttk.Label(self.stage_frame, text="Roller Area (m^2):").grid(row=i *␣
↪6 + 2, column=0, sticky="w") # Adjusted row
            ttk.Entry(self.stage_frame, textvariable=self.roller_areas[i]).
↪grid(row=i * 6 + 2, column=1) # Adjusted row
            ttk.Label(self.stage_frame, text="Juice Viscosity (Pa.s):").
↪grid(row=i * 6 + 3, column=0, sticky="w") # Adjusted row
            ttk.Entry(self.stage_frame, textvariable=self.juice_viscosities[i]).
↪grid(row=i * 6 + 3, column=1) # Adjusted row
            ttk.Label(self.stage_frame, text="Pressure Drop (Pa):").grid(row=i␣
↪* 6 + 4, column=0, sticky="w") # Adjusted row
            ttk.Entry(self.stage_frame, textvariable=self.pressure_drops[i]).
↪grid(row=i * 6 + 4, column=1) # Adjusted row
```

```
            ttk.Label(self.stage_frame, text="Imbibition Rate (kg):").
↪grid(row=i * 6 + 5, column=0, sticky="w") # Adjusted row
            ttk.Entry(self.stage_frame, textvariable=self.imbibition_rates[i]).
↪grid(row=i * 6 + 5, column=1) # Adjusted row

        # --- Run Button ---
        self.run_button = ttk.Button(master, text="Run Simulation",␣
↪command=self.run_simulation)
        self.run_button.grid(row=1, column=0, columnspan=2, pady=10)

        # --- Output Frame ---
        self.output_frame = ttk.LabelFrame(master, text="Results", padding=(10,␣
↪10))
        self.output_frame.grid(row=2, column=0, columnspan=2, padx=10, pady=10,␣
↪sticky="nsew")

        # --- Results Treeview ---
        self.tree = ttk.Treeview(self.output_frame)
        self.tree["columns"]=("Stage","Juice Mass", "Juice Pol", "Juice Brix",␣
↪"Bagasse Mass", "Bagasse Pol", "Bagasse Brix", "Extraction Efficiency",␣
↪"Darcy Number")
        self.tree.column("#0", width=0,  stretch=tk.NO)
        self.tree.column("Stage",anchor=tk.CENTER, width=80)
        self.tree.column("Juice Mass",anchor=tk.CENTER, width=100)
        self.tree.column("Juice Pol",anchor=tk.CENTER, width=80)
        self.tree.column("Juice Brix",anchor=tk.CENTER, width=80)
        self.tree.column("Bagasse Mass",anchor=tk.CENTER, width=100)
        self.tree.column("Bagasse Pol",anchor=tk.CENTER, width=80)
        self.tree.column("Bagasse Brix",anchor=tk.CENTER, width=80)
        self.tree.column("Extraction Efficiency",anchor=tk.CENTER, width=120)
        self.tree.column("Darcy Number",anchor=tk.CENTER, width=100)

        self.tree.heading("#0",text="",anchor=tk.CENTER)
        self.tree.heading("Stage",text="Stage",anchor=tk.CENTER)
        self.tree.heading("Juice Mass",text="Juice Mass (kg)",anchor=tk.CENTER)
        self.tree.heading("Juice Pol",text="Juice Pol (%)",anchor=tk.CENTER)
        self.tree.heading("Juice Brix",text="Juice Brix (%)",anchor=tk.CENTER)
        self.tree.heading("Bagasse Mass",text="Bagasse Mass (kg)",anchor=tk.
↪CENTER)
        self.tree.heading("Bagasse Pol",text="Bagasse Pol (%)",anchor=tk.CENTER)
        self.tree.heading("Bagasse Brix",text="Bagasse Brix (%)",anchor=tk.
↪CENTER)
        self.tree.heading("Extraction Efficiency",text="Extraction␣
↪Efficiency",anchor=tk.CENTER)
        self.tree.heading("Darcy Number",text="Darcy Number",anchor=tk.CENTER)
```

```
        self.tree.pack(expand=True, fill="both")

        # --- Plot Frame ---
        self.plot_frame = ttk.LabelFrame(master, text="Extraction Efficiency␣
↪Plot", padding=(10, 10))
        self.plot_frame.grid(row=3, column=0, columnspan=2, padx=10, pady=10,␣
↪sticky="nsew")

        self.fig, self.ax = plt.subplots(figsize=(8, 4)) # Create Figure and␣
↪Axes object
        self.canvas = FigureCanvasTkAgg(self.fig, master=self.plot_frame)
        self.canvas.get_tk_widget().pack(expand=True, fill="both")

    def run_simulation(self):
        try:
            # --- Get Input Values from GUI ---
            cane_mass = self.cane_mass.get()
            cane_pol = self.cane_pol.get()
            cane_brix = self.cane_brix.get()
            cane_preparation_index = self.cane_preparation_index.get()
            imbibition_pol = self.imbibition_pol.get()
            imbibition_brix = self.imbibition_brix.get()

            roller_configs = []
            imbibition_rates = []
            for i in range(self.num_stages):
                permeability =␣
↪adjust_permeability_based_on_preparation(cane_preparation_index)
                roller_configs.append({
                    'roller_gap': self.roller_gaps[i].get(),
                    'roller_surface_area': self.roller_areas[i].get(),
                    'permeability': permeability,
                    'juice_viscosity': self.juice_viscosities[i].get(),
                    'pressure_drop': self.pressure_drops[i].get()
                })
                imbibition_rates.append(self.imbibition_rates[i].get())

            # --- Run Simulation ---
            results_df = multi_roller_extraction(cane_mass, cane_pol,␣
↪cane_brix, roller_configs, imbibition_rates, imbibition_pol, imbibition_brix)

            # --- Display Results in Treeview ---
            for item in self.tree.get_children():
                self.tree.delete(item)

            for index, row in results_df.iterrows():
                self.tree.insert("", tk.END, values=(
```

```
                    row['Stage'],
                    f"{row['Juice Mass (kg)']:.2f}",
                    f"{row['Juice Pol (%)']:.2f}",
                    f"{row['Juice Brix (%)']:.2f}",
                    f"{row['Bagasse Mass (kg)']:.2f}",
                    f"{row['Bagasse Pol (%)']:.2f}",
                    f"{row['Bagasse Brix (%)']:.2f}",
                    f"{row['Extraction Efficiency']:.3f}",
                    f"{row['Darcy Number']:.2e}"
                ))

            # --- Plot Extraction Efficiency ---
            self.ax.clear()  # Clear the previous plot
            self.ax.plot(results_df['Stage'], results_df['Extraction␣
 ↪Efficiency'], marker='o')
            self.ax.set_xlabel("Stage")
            self.ax.set_ylabel("Extraction Efficiency")
            self.ax.set_title("Extraction Efficiency per Stage")
            self.ax.set_ylim(0, 1.05)  # Set y-axis limit to 0-1.05
            self.ax.grid(True)
            self.canvas.draw()  # Redraw the canvas

        except ValueError as e:
            messagebox.showerror("Error", str(e)) # Show errors to the user.
        except Exception as e:
            messagebox.showerror("Error", f"An unexpected error occurred:␣
 ↪{str(e)}")

# --- Main Execution ---
if __name__ == '__main__':
    root = tk.Tk()
    app = SugarcaneApp(root)
    root.mainloop()
```