Sudoku LP

Assignment task:

Formulate a Sudoku problem as an LP mathematical formula and then solve it with Python and MATLAB.

Problem description:

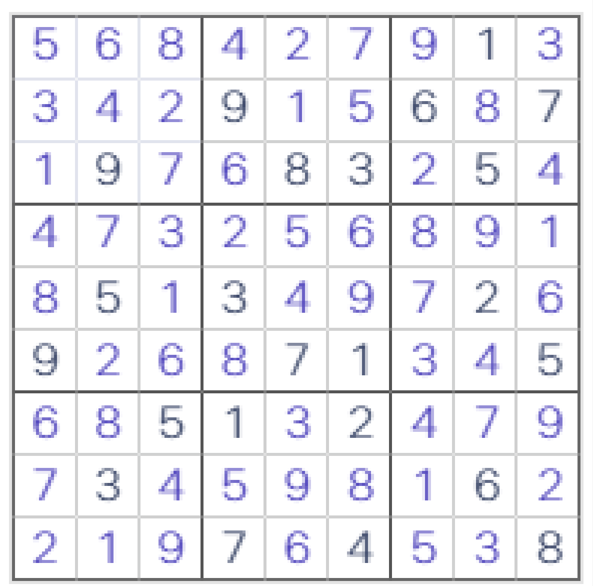
A Sudoku grid is a 9 × 9 matrix divided into 3 × 3 subgrids.
The rules for solving Sudoku are:

1. Each **row** must contain the digits **1 – 9** exactly once.
2. Each **column** must contain the digits **1 – 9** exactly once.
3. Each **3 × 3 subgrid** must contain the digits **1 – 9** exactly once.
4. Certain cells are **pre-filled as constraints**, and these values must remain unchanged.

The problem is modeled as a **binary optimization problem** where the solution is computed using LP solvers.

This problem is solved on a “extreme” difficulty sudoku problem:

Dark blue numbers represent the pre-filled constraints.



5	6	8	4	2	7	9	1	3
3	4	2	9	1	5	6	8	7
1	9	7	6	8	3	2	5	4
4	7	3	2	5	6	8	9	1
8	5	1	3	4	9	7	2	6
9	2	6	8	7	1	3	4	5
6	8	5	1	3	2	4	7	9
7	3	4	5	9	8	1	6	2
2	1	9	7	6	4	5	3	8

Source: Hard sudoku puzzles online

Python implementation:

1. **Initialization**:
   - A linear programming problem is created using `pulp.LpProblem` with a minimization goal (`pulp.LpMinimize`).
   - Binary decision variables `x[(i, j, k)]` are defined, where:
     - `i` and `j` are row and column indices (1–9),
     - `k` is the possible digit (1–9).

   ```
   x = pulp.LpVariable.dicts("x",
                             ((i, j, k) for i in range(1, 10)
                                         for j in range(1, 10)
                                         for k in range(1, 10)),
                             cat="Binary")
   ```

2. **Objective Function**:
   - The objective function is set to `0` since solving Sudoku doesn't require optimization, only feasibility.
3. **Constraints**:
   - **Cell Constraints**:
     - Each cell contains exactly one number (sum over `k` equals 1 for each `(i, j)`).

   ```
   for i in range(1, 10):
       for j in range(1, 10):
           prob += pulp.lpSum(x[(i, j, k)] for k in range(1, 10)) == 1
   ```

   - **Row Constraints**:
     - Each digit (1–9) appears exactly once in every row.

   ```
   for i in range(1, 10):
       for k in range(1, 10):
           prob += pulp.lpSum(x[(i, j, k)] for j in range(1, 10)) == 1
   ```

   - **Column Constraints**:
     - Each digit (1–9) appears exactly once in every column.

   ```
   for j in range(1, 10):
       for k in range(1, 10):
           prob += pulp.lpSum(x[(i, j, k)] for i in range(1, 10)) == 1
   ```

   - **Sub-grid Constraints**:

        - Each digit (1–9) appears exactly once in each 3×3 sub-grid.

```
for sub_i in range(0, 3):
    for sub_j in range(0, 3):
        for k in range(1, 10):
            prob += pulp.lpSum(x[(i, j, k)]
                               for i in range(1 + sub_i * 3, 4 + sub_i * 3)
                               for j in range(1 + sub_j * 3, 4 + sub_j * 3)) == 1
```

    - **Pre-filled Cells:**
        - Specific cells are pre-assigned values using the `prefilled_cells` dictionary, and these constraints are enforced.

```
prefilled_cells = {(1, 8): 1,
                   (2, 4): 9, (2, 7):6, (2,9):7,
                   (3,2):9, (3,5):8, (3,6):3, (3,8):5,
                   (5,2):5, (5,4):3, (5,8):2,
                   (6,1):9, (6,5):7, (6,6):1, (6,9):5,
                   (7,3):5, (7,4):1, (7,6):2,
                   (8,2):3, (8,8):6,
                   (9,4):7, (9,6):4, (9,9):8}

for (i, j), k in prefilled_cells.items():
    prob += x[(i, j, k)] == 1
```

4. **Solution**:
    - The `pulp.solve()` method is called to solve the LP problem.
    - The solution values are extracted from the decision variables, and a 9×9 Sudoku grid is created.
5. **Output**:
    - The final Sudoku solution is printed row by row.

```
[5, 6, 8, 4, 2, 7, 9, 1, 3]
[3, 4, 2, 9, 1, 5, 6, 8, 7]
[1, 9, 7, 6, 8, 3, 2, 5, 4]
[4, 7, 3, 2, 5, 6, 8, 9, 1]
[8, 5, 1, 3, 4, 9, 7, 2, 6]
[9, 2, 6, 8, 7, 1, 3, 4, 5]
[6, 8, 5, 1, 3, 2, 4, 7, 9]
[7, 3, 4, 5, 9, 8, 1, 6, 2]
[2, 1, 9, 7, 6, 4, 5, 3, 8]
```

MATLAB implementation:

1. **Initialization**
    - A 3D binary decision variable x is created to represent the Sudoku grid:
        - x(i, j, k) = 1 if the digit k is in cell (i, j), otherwise 0.

```
n = 9;
x = optimvar('x', n, n, n, 'Type', 'integer', 'LowerBound', 0,
'UpperBound', 1);
```

2. **Optimization Problem**
    - An optimization problem prob is initialized with no objective function:

```
prob = optimproblem;
```

3. **Constraints**
    - **Cell Constraints**:
      Each cell contains exactly one digit (sum over all possible digits equals

```
cell_constraint = sum(x, 3) == 1;
prob.Constraints.cell = cell_constraint;
```

- **Row Constraints**:
  Each row must contain each number exactly once:

```
for k = 1:n
    prob.Constraints.(['row_', num2str(k)]) = sum(x(:, :, k), 2) == 1;
end
```

- **Column Constraints**:
  Each column must contain each number exactly once:

```
for k = 1:n
    prob.Constraints.(['col_', num2str(k)]) = sum(x(:, :, k), 1)' == 1;
end
```

- **Sub-grid Constraints:**
  Each 3x3 sub-grid must contain each number exactly once:

```
for sub_i = 0:2
    for sub_j = 0:2
        for k = 1:n
            prob.Constraints.(['subgrid_', num2str(sub_i), '_',
num2str(sub_j), '_', num2str(k)]) = ...
                sum(sum(x(sub_i*3+1:sub_i*3+3, sub_j*3+1:sub_j*3+3, k)))
== 1;
        end
    end
end
```

4. **Pre-filled Cells**
   - The known values in the Sudoku grid are fixed using constraints:

```
prefilled_cells = [
    1, 8, 1;
    2, 4, 9; 2, 7, 6; 2, 9, 7;
    3, 2, 9; 3, 5, 8; 3, 6, 3; 3, 8, 5;
    5, 2, 5; 5, 4, 3; 5, 8, 2;
    6, 1, 9; 6, 5, 7; 6, 6, 1; 6, 9, 5;
    7, 3, 5; 7, 4, 1; 7, 6, 2;
    8, 2, 3; 8, 8, 6;
    9, 4, 7; 9, 6, 4; 9, 9, 8;
];
for i = 1:size(prefilled_cells, 1)
    row = prefilled_cells(i, 1);
    col = prefilled_cells(i, 2);
    digit = prefilled_cells(i, 3);
    prob.Constraints.(['prefilled_', num2str(i)]) = x(row, col, digit)
== 1;
end
```

5. **Solving the Problem**
   - The `intlinprog` solver is used to solve the feasibility problem:

```
intcon = 1:n*n*n;  % Indices of integer variables
f = zeros(n, n, n); % Objective function is zero
[x_sol, ~, exitflag] = solve(prob, 'Options', optimoptions('intlinprog',
'Display', 'off'));
```

6. **Extracting and Displaying the Solution**
   - The 3D solution variable `x_sol` is processed to extract the Sudoku grid:

```
if exitflag == 1
    sudoku_grid = zeros(n, n);
    for i = 1:n
        for j = 1:n
            for k = 1:n
                if x_sol.x(i, j, k) > 0.5  % Binary solution check
                    sudoku_grid(i, j) = k;
                end
            end
        end
    end
    disp('Solved Sudoku Grid:');
    disp(sudoku_grid);
else
    disp('No solution found or solver did not converge.');
end
```

7. **Output**

```
Solved Sudoku Grid:
     5     6     8     4     2     7     9     1     3
     3     4     2     9     1     5     6     8     7
     1     9     7     6     8     3     2     5     4
     4     7     3     2     5     6     8     9     1
     8     5     1     3     4     9     7     2     6
     9     2     6     8     7     1     3     4     5
     6     8     5     1     3     2     4     7     9
     7     3     4     5     9     8     1     6     2
     2     1     9     7     6     4     5     3     8
```

Comparison

- **Python (PuLP)**:
  - Simple and clear implementation using `LpProblem` and binary decision variables.
  - Solves the problem as a feasibility LP problem.
  - Output is processed and displayed in a 2D grid format.
- **MATLAB**:
  - Uses a structured `optimproblem` framework for defining constraints and solving with `intlinprog`.
  - Modular constraint definitions make the implementation clean and extensible.
  - Directly integrates integer programming tools for optimization problems.