Mathematical Walkthrough of the Matrix Filler Algorithm

The objective of this algorithm is to fill missing values (`NaN`) in a matrix so that each row and column sum equals a specified target. The mathematical problem is formulated as a bounded linear least squares system:

$$\min_x \ \|Ax - b\|_2^2 \quad \text{subject to} \quad \ell \leq x \leq u,$$

where

- $x \in \mathbb{R}^K$ are the unknown cell values (one variable per missing cell),
- $A \in \mathbb{R}^{(n_r+n_c)\times K}$ encodes which variables belong to each row and column,
- $b \in \mathbb{R}^{n_r+n_c}$ are the target totals (row and column sums) after subtracting known values,
- and the bounds are typically $\ell = 0, \ u = +\infty$ to enforce nonnegativity.

Example: 4×4 Matrix

Consider the grid

$$G = \begin{bmatrix} 2 & \text{NaN} & \text{NaN} & 3 \\ \text{NaN} & 1 & \text{NaN} & \text{NaN} \\ \text{NaN} & \text{NaN} & 4 & \text{NaN} \\ 3 & \text{NaN} & \text{NaN} & 1 \end{bmatrix}, \quad r = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 10 \end{bmatrix}, \quad c = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 10 \end{bmatrix}.$$

There are $K = 10$ unknown cells, listed in row-major order:

$$U = \{(0,1),(0,2),(1,0),(1,2),(1,3),(2,0),(2,1),(2,3),(3,1),(3,2)\}.$$

We associate one variable per missing cell:

$$x = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{bmatrix}^\top.$$

Building the Linear System

Row constraints. For each row i, we require that the sum of all entries equals its target:

$$\sum_{j=0}^{n_c-1} G_{ij} = r_i.$$

Known values are moved to the right-hand side:

$$\sum_{(i,j)\in U} G_{ij} = r_i - \sum_{\substack{j \\ G_{ij}\ \text{known}}} G_{ij}.$$

This contributes one equation to A and b. Each variable x_k that belongs to that row receives a coefficient of 1 in A.

Column constraints. Similarly, for each column j:

$$\sum_{i=0}^{n_r-1} G_{ij} = c_j$$

This produces one additional equation per column.

Computing b

$$\sum_{\text{NaNs in row } i} x_k = r_i - \sum_{\text{knowns in row } i} \text{known},$$

$$\sum_{\text{NaNs in col } j} x_k = c_j - \sum_{\text{knowns in col } j} \text{known}.$$

Known row and column sums are:

$$\text{Row knowns: } [5,\ 1,\ 4,\ 4] \Rightarrow b_{\text{rows}} = [5,\ 9,\ 6,\ 6],$$

$$\text{Column knowns: } [5,\ 1,\ 4,\ 4] \Rightarrow b_{\text{cols}} = [5,\ 9,\ 6,\ 6].$$

Stacking them yields

$$b = \begin{bmatrix} 5 \\ 9 \\ 6 \\ 6 \\ 5 \\ 9 \\ 6 \\ 6 \end{bmatrix}.$$

Constructing A

Each row of A corresponds to one equation (a row sum or column sum). Each column of A corresponds to one variable x_k. If a variable belongs to that row or column, its coefficient is 1; otherwise 0.

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Notice that each column of A contains exactly two 1's — one for its row and one for its column equation.

Solving via SciPy

We solve

$$\min_{x \geq 0} \|Ax - b\|_2^2.$$

SciPy's `lsq_linear(A, b, bounds=(0, +inf))` uses the Trust Region Reflective algorithm to find the nonnegative least-squares solution.

$$x = \begin{bmatrix} 3.25 & 1.75 & 3.25 & 2.00 & 3.75 & 1.75 & 2.00 & 2.25 & 3.75 & 2.25 \end{bmatrix}^\top.$$

The solver reports `success = True` and the unconstrained solution is optimal, meaning all equations can be satisfied exactly.

Filling the Matrix

Each variable x_k is written back to its corresponding location in G, yielding the filled grid:

$$\hat{G} = \begin{bmatrix} 2 & 3.25 & 1.75 & 3.00 \\ 3.25 & 1 & 2.00 & 3.75 \\ 1.75 & 2.00 & 4 & 2.25 \\ 3.00 & 3.75 & 2.25 & 1 \end{bmatrix}.$$

Every row and column now sums to 10, and all entries are nonnegative:

$$\text{Row sums: } [10, 10, 10, 10], \quad \text{Column sums: } [10, 10, 10, 10].$$

Interpretation

- Each element of b represents the *remaining total* to be filled after subtracting known entries.
- Each column of A corresponds to a missing cell, with 1's marking the equations (row and column) it participates in.
- The vector x provides the optimal values for all missing cells.
- Using least squares ensures that even if row and column targets conflict, the algorithm finds the closest possible fit while keeping values within bounds.