Fitting Tabulated Data Using SVD

Student Laboratory Notes and Step-by-Step Guide

Numerical Methods in Physics • AGH University of Krakow

PAGE 1 From data points to the matrix we must build

1. What is our goal? We must write our own C++ program to find a polynomial that approximates known data values. The official input function is

$$\boxed{y(x) = x\cos x, \qquad 0 \le x \le 3\pi.}$$

We do not know the polynomial coefficients in advance. We choose a polynomial with n unknown coefficients and degree $n-1$:

$$\boxed{p(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}.}$$

We use **SVD** to calculate the coefficients from the given values. A fit need not pass through every data point.

2. How do we generate our official data?

For the first laboratory experiment, we use $m = 8$ **data points**, $n = 6$ **coefficients**, $x_a = 0$, and $x_b = 3\pi$. We generate equally spaced positions and function values:

$$x_i = x_a + i\frac{x_b - x_a}{m-1}, \quad y_i = x_i\cos(x_i), \quad i = 0, \ldots, m-1.$$

Here m counts observations, while n counts unknown polynomial coefficients. Our first model has degree 5; it does not have degree 6.

3. Where does the design matrix come from?

For each data point we write $p(x_i) \approx y_i$. Collecting these m equations gives

$$\boxed{A\mathbf{c} \approx \mathbf{y}, \qquad A_{ij} = x_i^j, \qquad A \in \mathbb{R}^{m\times n}.}$$

The first column of A is all ones; the next contains x_i; then x_i^2, and so on. The unknown vector $\mathbf{c}$ has n elements and the data vector $\mathbf{y}$ has m elements.

A small example: for $x = (0, 1, 2)$ and a line $p(x) = c_0 + c_1 x$,

$$\underbrace{\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}}_{A\in\mathbb{R}^{3\times 2}} \underbrace{\begin{pmatrix} c_0 \\ c_1 \end{pmatrix}}_{\mathbf{c}} \approx \underbrace{\begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}}_{\mathbf{y}}.$$

Because $m > n$, this system is generally *overdetermined*. We do not demand exact equality. Instead, we minimize the total squared residual:

$$\boxed{\min_{\mathbf{c}} \|A\mathbf{c} - \mathbf{y}\|_2^2.}$$

Examples E01–E02 help us construct A and verify a simple least-squares fit before we work on the official data.

PAGE 2 How SVD gives us the polynomial coefficients

4. What does singular value decomposition mean?

For a rectangular matrix A with $m \geq n$, we use the *thin* decomposition

$$\boxed{A = U\Sigma V^T, \qquad U : m \times n, \quad \Sigma : n \times n, \quad V^T : n \times n.}$$

The columns $\mathbf{u}_k$ of U are normalized left singular vectors, the columns $\mathbf{v}_k$ of V are normalized right singular vectors, and the diagonal entries σ_k of Σ are *singular values*. They are nonnegative and normally appear in descending order:

$$\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{n-1} \geq 0, \qquad U^T U = I_n, \quad V^T V = I_n.$$

For the official first experiment, the matrix sizes are $A : 8 \times 6$, $U : 8 \times 6$, $\Sigma : 6 \times 6$, and $V^T : 6 \times 6$.

5. How do we check that SVD worked?

Every matrix entry can be reconstructed by adding singular contributions:

$$\boxed{A = \sum_{k=0}^{n-1} \sigma_k \mathbf{u}_k \mathbf{v}_k^T, \qquad A_{ij} = \sum_{k=0}^{n-1} \sigma_k U_{ik} (V^T)_{kj}.}$$

We construct the matrix A_{rec} from the computed singular vectors and compare it against our saved original A. We may measure the relative error $\|A - A_{\text{rec}}\|_F / \|A\|_F$. A small value means the decomposition reconstructs the matrix accurately; it does not by itself prove the fitted curve is accurate.

6. How do we obtain the fit coefficients?

For full column rank, the SVD least-squares formula is

$$\boxed{\mathbf{c} = V\Sigma^{-1}U^T\mathbf{y} = \sum_{k=0}^{n-1} \frac{\mathbf{u}_k^T \mathbf{y}}{\sigma_k} \mathbf{v}_k.}$$

We interpret the formula as four actions:

1. We take the dot product $\alpha_k = \mathbf{u}_k^T \mathbf{y}$ (projection of data onto direction k).
2. We divide by the corresponding nonzero singular value σ_k.
3. We scale the right singular vector $\mathbf{v}_k$ by α_k / σ_k.
4. We add the contributions to obtain all polynomial coefficients c_j.

If a singular value is zero or numerically negligible, we use an appropriate pseudoinverse convention rather than dividing by zero. A very small singular value may amplify numerical errors.

Teaching check: E03 reconstructs an exact 2×2 matrix; E04 calls LAPACKE for an unrelated 4×2 matrix; E05 fits three points and recovers $c_0 = 7/6$, $c_1 = 1/2$. These examples are independent of our official assessed data.

PAGE 3 What exactly must we implement in C++?

7. Our step-by-step programming workflow

1. **Set the parameters.** We define $m = 8$, $n = 6$, $x_a = 0$, and $x_b = 3\pi$. We use floating-point numbers of type `double` and store our matrices and vectors consistently.
2. **Generate the data.** We compute all equally spaced x_i and all $y_i = x_i \cos x_i$. We inspect the first/last nodes to make sure the interval and spacing are correct.
3. **Build the design matrix.** We fill $A_{ij} = x_i^j$ and *write the matrix to a file*, as required. In row-major one-dimensional storage,
$$\boxed{A_{ij} \longleftrightarrow \texttt{a[i*n+j]}.}$$
4. **Preserve the original matrix.** We keep an unchanged copy of A because `LAPACKE_dgesvd()` may overwrite its input array.
5. **Allocate thin SVD arrays.** With $m \geq n$, we prepare U with mn entries, V^T with n^2 entries, the n singular values, and an appropriate `superb` buffer.
6. **Call LAPACKE.** We call `LAPACKE_dgesvd()` using `LAPACK_ROW_MAJOR` and `'S','S'`, with `lda=n`, `ldu=n`, `ldvt=n`. We check the returned `info` value.
7. **Save singular values.** We write all σ_k to a readable file; we inspect their order and magnitudes.
8. **Reconstruct the matrix.** We evaluate the rank-one sum for each entry A_{ij} and compare with the saved original matrix. The key array indices are
$$\boxed{(\mathbf{u}_k)_i \longleftrightarrow \texttt{u[i*n+k]}, \qquad (\mathbf{v}_k)_j \longleftrightarrow \texttt{vt[k*n+j]}.}$$
9. **Calculate polynomial coefficients.** We project the data onto the $\mathbf{u}_k$, divide by retained nonzero singular values, and combine the $\mathbf{v}_k$. We save all c_j in a file.
10. **Prepare fitted-curve data.** We evaluate $p(x) = \sum_{j=0}^{n-1} c_j x^j$ on a dense plotting grid. We may use Horner's rule to reduce unnecessary power evaluations.

8. What should we verify?

We verify that the stored A has the intended powers and dimensions, that LAPACKE succeeded, and that $A_{\text{rec}} \approx A$. We also evaluate the fit residuals $r_i = p(x_i) - y_i$ at the original points. We must not confuse a small reconstruction error with a small fitting error.

The five complete C++ programs teach the small building blocks. We must combine them in our *own* official laboratory program; a complete assessed implementation is intentionally not supplied.

PAGE 4 Homework, figures, report, and checks

9. We repeat the full procedure for three configurations

Experiment	Data points m	Coefficients n	Degree
Main laboratory	8	6	5
Homework A	30	10	9
Homework B	30	20	19

The domain $[0, 3\pi]$ and the input function $y(x) = x\cos x$ remain unchanged. The official assignment asks us to repeat its computational procedure for the two homework cases.

10. How do we prepare the required report plots?

We prepare a **separate figure for each** (m, n) pair. In each figure we show the original input points and our fitted polynomial. Evaluating the polynomial on many additional x values helps us see what happens between the original data points. We may also overlay $x\cos x$ to clarify the comparison.

The example panel $(m, n) = (30, 15)$ shown in the official PDF is an *illustration*; it is not a substitute for either required homework configuration.

11. What should we discuss?

- We compare the fitted curves with the tabulated input values for all three experiments.
- We comment on residuals at the nodes and differences between nodes.
- We explain why a higher-degree polynomial may behave differently and why increasing degree need not guarantee a stable approximation across the whole interval.
- We consider the influence of small singular values and the wide range of powers x_i^j on numerical conditioning.
- We distinguish numerical SVD correctness from physical/model fitting quality.

12. Our final checklist

1. We have generated data and saved the design matrix for the required configurations.
2. We have calculated and saved singular values and polynomial coefficients.
3. We have reconstructed the matrix and checked its consistency with the original.
4. We have prepared **three distinct fitting plots** and written a clear comparison of the results.
5. Our explanations and implementation follow the original assignment, while E01–E05 serve only as small learning examples (E06 is optional).

We connect polynomial fitting with linear least squares, SVD, singular vectors, numerical programming and scientific interpretation. We should understand and verify each operation, rather than treating a fitted graph as proof that our code is correct.

Reference: Tomasz Chwiej, *Fitting tabulated data using singular values decomposition (SVD)*, 14 April 2026.