Curve fitting, data regression and optimization

Ivo Roghair, Martin van Sint Annaland

Chemical Process Intensification,
Eindhoven University of Technology

Today's outline

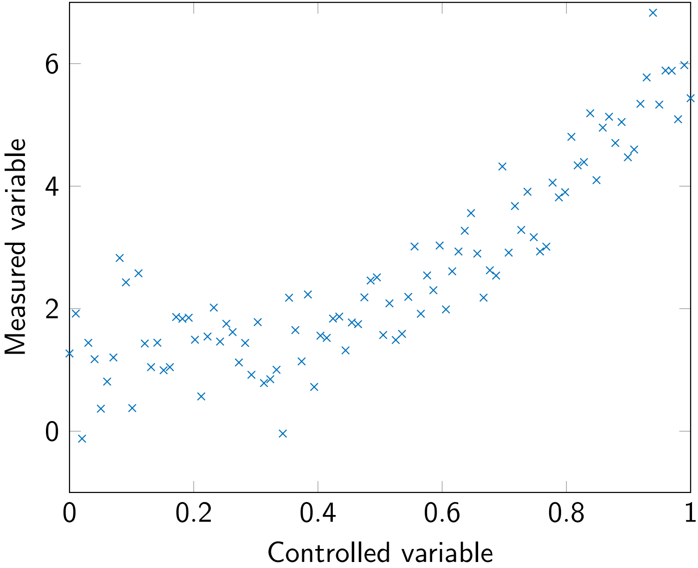
Today's outline

1. Introduction
2. Curve fitting
3. Regression
4. Fitting numerical models
5. Optimization
6. Linear programming
7. Summary

Overview

- We are going to fit measurements to models today
- You will also learn what R^2 actually means
- We get introduced to constrained and unconstrained optimization.
- We will use the simplex method to solve linear programming problems (LP)

Fitting models to data



How to fit a model to the data?

We would like to fit the following model to the data:

$$\hat{y} = a_1 + a_2 x + a_3 x^2 + a_4 x^3$$

How to fit a model to the data?

We would like to fit the following model to the data:

$$\hat{y} = a_1 + a_2 x + a_3 x^2 + a_4 x^3$$

First step: If we have N data points, we could write the model as the product of a matrix and a vector:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & \vdots & \\ 1 & x_N & x_N^2 & x_N^3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

$$\hat{y} = Xa$$

X is called the design matrix and a are the fit parameters.

Residuals

Second step: work out the residuals for each data point:

$$d_i = (y_i - \hat{y}_i)$$

Residuals

Second step: work out the residuals for each data point:

$$d_i = (y_i - \hat{y}_i)$$

Third step: work out the sum of squares of the residuals:

$$\mathrm{SSE} = \sum_i d_i^2 = \sum_i (y_i - \hat{y}_i)$$

This can be written using the dot-product operation:

$$\mathrm{SSE} = \sum_i d_i^2 = d \cdot d = d^T \cdot d = (y_i - \hat{y}_i)^T \cdot (y_i - \hat{y}_i)$$

Minimizing the sum of squares

Choose the parameter vector such that the sum of squares of the residuals is minimized; the partial derivative with respect to each parameter should be set to zero:

$$\frac{\partial}{\partial a_i}\left[\left(y-(Xa)^T\right)(y-Xa)\right]$$

Minimizing the sum of squares

Choose the parameter vector such that the sum of squares of the residuals is minimized; the partial derivative with respect to each parameter should be set to zero:

$$\frac{\partial}{\partial a_i}\left[\left(y-(Xa)^T\right)(y-Xa)\right]$$

In Matlab, we can solve our linear system $\hat{Y} = Xa$ simply by running `a = X\y`.

Minimizing the sum of squares

Choose the parameter vector such that the sum of squares of the residuals is minimized; the partial derivative with respect to each parameter should be set to zero:

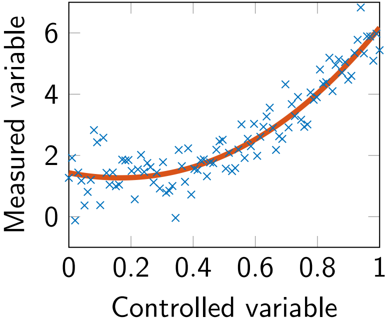
$$\frac{\partial}{\partial a_i}\left[\left(y-(Xa)^T\right)(y-Xa)\right]$$

In Matlab, we can solve our linear system $\hat{Y} = Xa$ simply by running `a = X\y`.

- If there are more data points ($N > 4$), we can write an analogue, but maybe a consistent solution does not exist (the system is over specified).
- However, matlab will find values for the vector a which minimize $\left\|y - aX\right\|^2$, so i.e. a least squares fit.

Fitting our problem: Matlab solver

```
N=length(x);
X(:,1) = ones(N,1);
X(:,2) = x;
X(:,3) = x.^2;
X(:,4) = x.^3;
A =X\y;
```



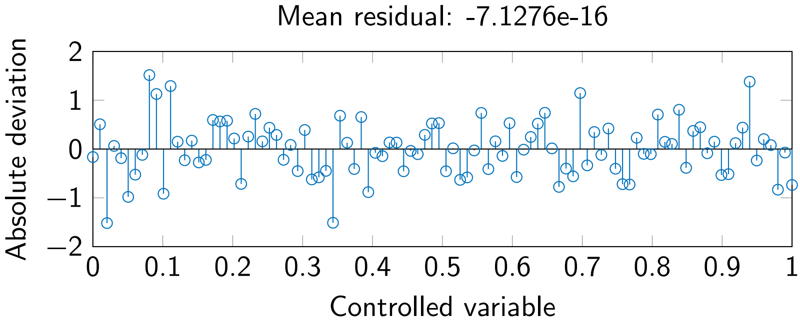
Fitting our problem: Matlab curve-fitting toolbox

- Start the toolbox: `cftool`
- Choose the dataset (x and y data)
- Choose the interpolant type (polynomial, exponential, ..., custom)
- Get the coefficients (save to workspace or write them down)

Fitting our problem: Excel

- Create a column with the independent and dependent data series (x and y)
- Create a column that computes $\hat{y}$, keeping the coefficients as separate cells
- Compute the sum of squares of the residuals (another column, sum the results)
- Use the solver to minimize this sum, modifying the coefficient cells
- Note: regression in Excel + display equation is dangerous if you choose the 'line' plot (use scatter if you can)

How good is the model?



- For a model to make sense the data points should be scattered randomly around the model predictions, the mean of the residuals d should be zero: $d_i = (y_i - \hat{y}_i)$
- It's always good to check if the residuals are not correlated with the measured values, if that is the case, it can indicate that your model is wrong.

Regression coefficients

- Variance measured in the data (y) is:

$$\sigma_y^2 = \frac{1}{N}\sum_i (y_i - \overline{y})^2$$

- Variance of the residuals is:

$$\sigma_{\text{error}}^2 = \frac{1}{N}\sum_i (d_i)^2$$

- Variance in the model is:

$$\sigma_{\text{model}}^2 = \frac{1}{N}\sum_i \left(\hat{y}_i - \overline{\hat{y}}\right)^2$$

Regression coefficients

Given that the error is uncorrelated we can state that:

$$\sigma_y^2 = \sigma_{\text{error}}^2 + \sigma_{\text{model}}^2$$

$$R^2 = \frac{\sigma_{\text{model}}^2}{\sigma_y^2} = 1 - \frac{\sigma_{\text{error}}^2}{\sigma_y^2}$$

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

- SSE: Sum of errors squared
- SST: Total sum of squares (model)
- SSR: Sum of squares (data)

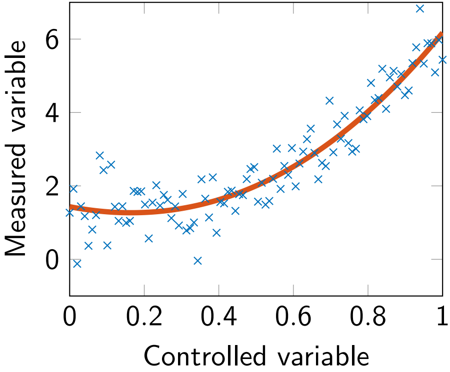
Regression coefficients

- An uncorrellated error ($\overline{d} \rightarrow 0$) means that SSE, SST and SSR will have χ^2 -distributions and the ratios will have an F-distribution. If SSR/SSE is large, the model is good!
- There is a chance that the model is rubbish, but that SSR/SSE will yield a good value, Analysis of Variance (ANOVA) will be a good tool to calculate the probability of such a thing happening!

Back to the example

The statistics:

	Value
N	100
SSE	32.042
SST	896.907
SSR	928.950
R^2	0.964



Dynamic fitting of non-linear equations: lsqnonlin

You may encounter situations where the model data is slightly more complicated to obtain (e.g. a numerical model based on ODEs where coefficients are unknown), or you want to perform fitting of multiple functions/coefficients, or just want to automate things via scripts. Matlab's Optimization toolbox gives access to a powerful function `lsqnonlin`, least-squares non-linear optimization.

General use of lsqnonlin

```
k = lsqnonlin(fun,k0,lb,ub,options)
```

- `fun` is a function handle to the fit criterium (e.g. `@myFitCrit`). The fit criterium function `myFitCrit` should return the residuals vector, e.g. $d_i = (y_i - \hat{y}_i)$. Here, y_i would again be the measurement data and $\hat{y}$ the solution computed by a model.
- `k0` is the initial guess for the fitting coefficient (or: array of initial guesses when fitting multiple coefficients)
- `lb` and `ub` are the lower and upper boundaries for `k0`. These should both be the size of the `k0`-array.
- `options` are some fitting options, for more fine-grained control on the fit procedure. Use e.g. `options = optimset('TolX',1.0E-6,'MaxFunEvals',1000);` to create an options object, or leave it empty (`options = []`).

Example use of lsqnonlin

We have experimental data stored in the file `tudataset1.mat`, containing T and U data. We want to fit a model with coefficients k_1 and k_2 with the following structure:

$$\frac{du}{dt} = -k_1 u + k_2$$

Example use of lsqnonlin

We have experimental data stored in the file `tudataset1.mat`, containing T and U data. We want to fit a model with coefficients k_1 and k_2 with the following structure:

$$\frac{du}{dt} = -k_1 u + k_2$$

- First, we need to create a function that contains the ODE:

```
function dudt = simpleode(t,u,k);
dudt = -k(1)*u + k(2);
```

  Note that we supply a vector `k`, containing both coefficients for fitting

Example use of lsqnonlin

We have experimental data stored in the file `tudataset1.mat`, containing T and U data. We want to fit a model with coefficients k_1 and k_2 with the following structure:

$$\frac{du}{dt} = -k_1 u + k_2$$

- First, we need to create a function that contains the ODE:

```
function dudt = simpleode(t,u,k);
dudt = -k(1)*u + k(2);
```

  Note that we supply a vector `k`, containing both coefficients for fitting
- We create a fit criterium function:

```
function err = fitcrit(ke,T,U,U0)
[t,ue] = ode45(@simpleode,T,U0,[],ke);
err = (ue-U);
```

Example use of lsqnonlin

Now let's make a script that uses `lsqnonlin` to yield k-values fitted to our dataset:

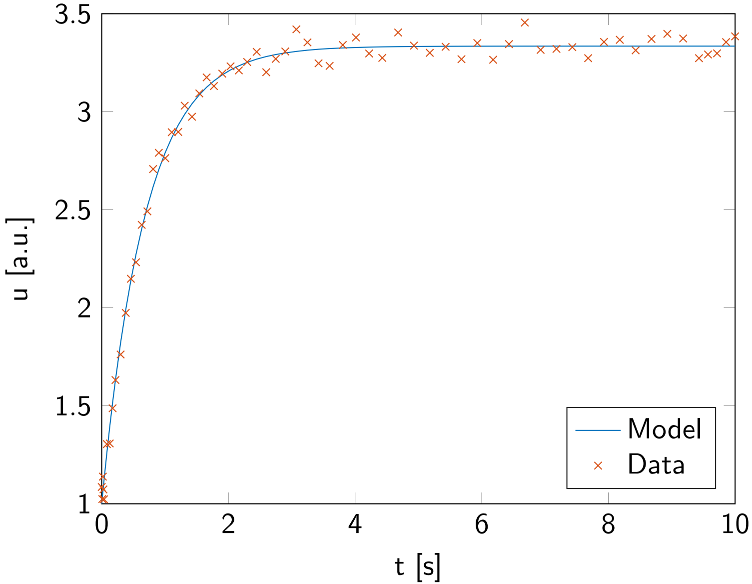
Example use of lsqnonlin

Now let's make a script that uses `lsqnonlin` to yield k-values fitted to our dataset:

```
% initial value
U0 = 1.00;
% initial guesses for model parameters
k0 = [1.00 1.00];
% lower and upper bounds for model parameters
LB = [0.00 0.00];
UB = [Inf Inf ];
% Perform nonlinear least squares fit
options = optimset('TolX',1.0E-6,'MaxFunEvals',1000);
[ke,RESNORM,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA,JACOBIAN]
    = lsqnonlin(@fitcrit,k0,LB,UB,options,T,U,U0);
```

Our fitted coefficients are stored in `ke`. Note that we get a lot more data back that allows to check the fitting results in more detail.

Example use of lsqnonlin



Postprocessing of results

The data returned by `lsqnonlin` can be used to obtain the 95% confidence intervals. Recall the command:

```
[ke,RESNORM,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA,JACOBIAN]
    = lsqnonlin(@fitcrit,k0,LB,UB,options,T,U,U0);
```

Using the residuals and Jacobian we can use `nlparci` to get the confidence bounds:

```
cflim = nlparci(ke, RESIDUAL, JACOBIAN);

clc
disp('model parameters and confidence limits');
T = table;
T.ke = ke';
T.LowerCI = cflim(:,1);
T.UpperCI = cflim(:,2)
```

Today's outline

What is optimization?

Optimization is minimization or maximization of an objective function (also called a performance index or goal function) that may be subject to certain constraints.

- $\min f(x)$: Goal function
- $g(x) = 0$: Equality constraints
- $h(x) \geq 0$: Inequality constraints

Optimization Spectrum

Problem	Method	Solvers
LP	Simplex method Barrier methods	Linprog (Matlab) CPLEX (GAMS, AIMMS, AMPL, OPB)
NLP QP	Lagrange multiplier method Successive linear programming Quadratic programming	Fminsearch/fmincon (Matlab) MINOS (GAMS, AMPL) CONOPT (GAMS)
MIP MILP MINLP MIQP	Branch and bound Dynamic programming Generalized Benders decomposition Outer approximation method Disjunctive programming	Bintprog (Matlab) DICOPT (GAMS) BARON (GAMS)

Factors of concern

- Continuity of the functions
- Convexity of the functions
- Global versus local optima
- Constrained versus unconstrained optima

Linear programming

In linear programming the objective function and the constraints are linear functions.

Linear programming

In linear programming the objective function and the constraints are linear functions.

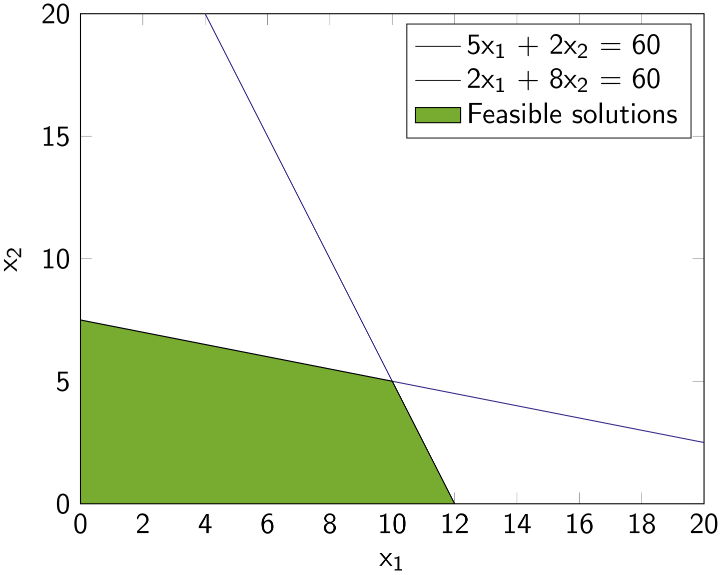
For example:

$\max z = f(x_1, x_2) = 40x_1 + 88x_2$
s.t. (subject to)
$2x_1 + 8x_2 \leq 60$
$5x_1 + 2x_2 \leq 60$
$x_1 \geq 0$
$x_2 \geq 0$

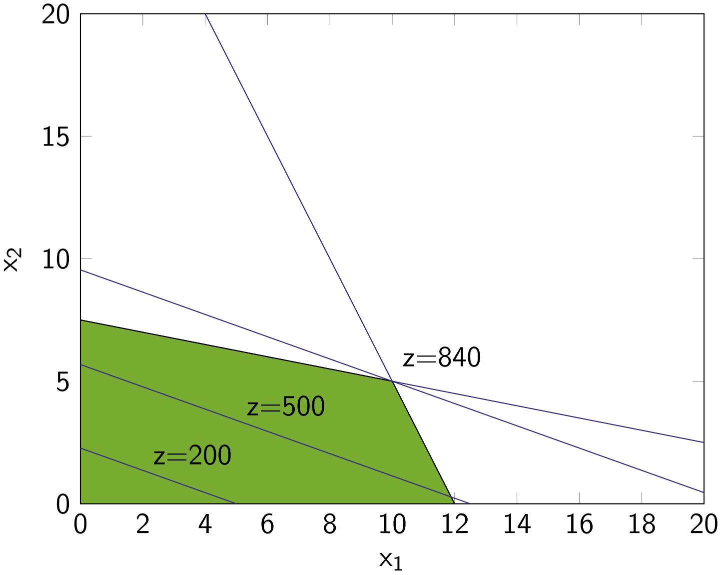
If the constraints are satisfied, but the objective function is not maximized/minimized we speak of a feasible solution.

If also the objective function is maximized/minimized, we speak of an optimal solution!

Plotting the constraints



Plotting the constraints



Normal form of an LP problem

$$
\begin{aligned}
&\max z = f(x_1, x_2) = 40x_1 + 88x_2 \\
&\text{s.t.} \\
&2x_1 + 8x_2 \leq 60 \\
&5x_1 + 2x_2 \leq 60 \\
&x_1 \geq 0 \\
&x_2 \geq 0
\end{aligned}
$$

Normal form of an LP problem

$$\max z = f(x_1, x_2) = 40x_1 + 88x_2$$
s.t.
$$2x_1 + 8x_2 \leq 60$$
$$5x_1 + 2x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

$$\max z = f(x) = 40x_1 + 88x_2$$
s.t.
$$2x_1 + 8x_2 + x_3 = 60$$
$$5x_1 + 2x_2 + x_4 = 60$$
$$x_i \geq 0 \qquad i \in 1, 2, 3, 4$$

x_3 and x_4 are called slack variables, they are non auxiliary variables introduced for the purpose of converting inequalities in to equalities

The simplex method

We can formulate our earlier example to the normal form and consider it as the following augmented matrix with $T_0 = [z \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad b]$:

$$T_0 = \begin{bmatrix} 1 & -40 & -88 & 0 & 0 & 0 \\ 0 & 2 & 8 & 1 & 0 & 60 \\ 0 & 5 & 2 & 0 & 1 & 60 \end{bmatrix}$$

This matrix is called the (initial) simplex table. Each simplex table has two kinds of variables, the basic variables (columns having only one nonzero entry) and the nonbasic variables

The simplex method

$$T_0 = \begin{bmatrix} 1 & -40 & -88 & 0 & 0 & 0 \\ 0 & 2 & 8 & 1 & 0 & 60 \\ 0 & 5 & 2 & 0 & 1 & 60 \end{bmatrix}$$

Every simplex table has a feasible solution. It can be obtained by setting the nonbasic variables to zero: $x_1 = 0$, $x_2 = 0$, $x_3 = 60/1$, $x_4 = 60/1$, $z = 0$.

The optimal solution?

- The optimal solution is now obtained stepwise by pivoting in such way that z reaches a maximum.
- The big question is, how to choose your pivot equation ...

Step 1: Selection of the pivot column

Select as the column of the pivot, the first column with a negative entry in Row 1. In our example, that's column 2 (-40)

$$T_0 = \begin{bmatrix} 1 & -40 & -88 & 0 & 0 & 0 \\ 0 & 2 & 8 & 1 & 0 & 60 \\ 0 & 5 & 2 & 0 & 1 & 60 \end{bmatrix}$$

Step 2: Selection of the pivot row

Divide the right sides by the corresponding column entries of the selected pivot column. In our example that is $60/2 = 30$ and $60/5 = 12$.

$$T_0 = \begin{bmatrix} 1 & -40 & -88 & 0 & 0 & 0 \\ 0 & 2 & 8 & 1 & 0 & 60 \\ 0 & 5 & 2 & 0 & 1 & 60 \end{bmatrix}$$

Take as the pivot equation the equation that gives the smallest quotient, so $60/5$.

Step 3: Elimination by row operations

- Row 1 = Row 1 + 8 * Row 3
- Row 2 = Row 2 - 0.4 * Row 3

$$T_1 = \begin{bmatrix} 1 & 0 & -72 & 0 & 8 & 480 \\ 0 & 0 & 7.2 & 1 & -0.4 & 36 \\ 0 & 5 & 2 & 0 & 1 & 60 \end{bmatrix}$$

The basic variables are now x_1, x_3 and the nonbasic variables are x_2, x_4. Setting the nonbasic variables to zero will give a new feasible solution: $x_1 = 60/5$, $x_2 = 0$, $x_3 = 36/1$, $x_4 = 0$, $z = 480$.

The simplex method

- We moved from $z = 0$ to $z = 480$. The reason for the increase is because we eliminated a negative term from the eqation, so: elimination should only be applied to negative entries in Row 1, but no others.
- Although we found a feasible solution, we did not find the optimal solution yet (the entry of -72 in our simplex table) $\longrightarrow$ repeat step 1 to 3.

The simplex method

Another iteration is required:

- Step 1: Select column 3
- Step 2: $36/7.2 = 5$ and $60/2 = 30 \longrightarrow$ select 7.2 as the pivot
- Elimination by row operations:
    - Row 1 = Row 1 + 10*Row 2
    - Row 3 = Row 3 - (2/7.2)*Row 2

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & 10 & 4 & 840 \\ 0 & 0 & 7.2 & 1 & -0.4 & 36 \\ 0 & 5 & 0 & -1/36 & 1/0.9 & 50 \end{bmatrix}$$

- The basic feasible solution: $x_1 = 50/5$, $x_2 = 36/7.2$, $x_3 = 0$, $x_4 = 0$, $z = 840$ (no more negative entries: so this solution is also the optimal solution)

Using Matlab for LP problems

We are going to solve the following LP problem:

$\min f(x) = -5x_1 - 4x_2 - 6x_3$

s.t.

$x_1 - x_2 + x_3 \leq 20$

$3x_1 + 2x_2 + 4x_3 \leq 42$

$3x_1 + 2x_2 \leq 30$

$x_1 \geq 0$

$x_2 \geq 0$

$x_3 \geq 0$

Using the function `linprog`:

```
f = [-5; -4; -6]
A = [1 -1 1; 3 2 4; 3 2 0];
b = [20; 42; 30];
lb = zeros(3,1);
[x,fval,exitflag,output,
    lambda]
= linprog(f,A,b,[],[],lb);
```

Gives:

```
x = 0.00 15.00 3.00
lambda.ineqlin = 0 1.50
    0.50
lambda.lower = 1.00 0 0
```

Summary

- Curve fitting: Manual procedures for polynomial fitting in Matlab and Excel
- Curve fitting: Matlab's curve-fitting toolbox
- Curve fitting: Matlab's non-linear least-squares solver `lsqnonlin`
- Optimization: An introduction to the Simplex method
- Optimization: Use of the `linprog` solver