Curve fitting, regression and optimization

Dr.ir. Ivo Roghair, Prof.dr.ir. Martin van Sint Annaland

Chemical Process Intensification group
Eindhoven University of Technology

Numerical Methods (6BER03), 2024-2025

Today's outline

- Introduction
- Curve fitting
- Regression
- Fitting numerical models
- Optimization
- Linear programming
- Summary

Today's outline

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)

Today's outline

- Introduction
- Curve fitting
- Regression
- Fitting numerical models
- Optimization
- Linear programming
- Summary

Let's do an 'experiment' to gather data

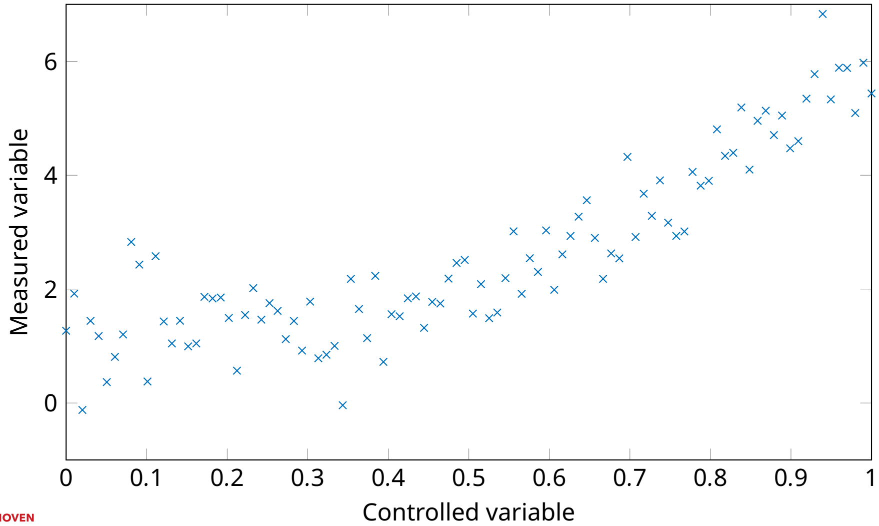
```python
def generate_random_data(N=101,p=[1,1/3,1.5,3.5],draw=False):
    # Generate linear space of control points
    # N - Number of data points
    # p - Coefficients of polynomial
    x = np.linspace(0, 1, N) # Points (independent variable)

    # Generate 'measurement values' with errors following a normal distribution
    # Initialize randomizer
    pd = norm(loc=0, scale=0.1)
    # Add scatter data to the polynomial
    y = np.polyval(p,x) + pd.rvs(size=N)

    # Plot the generated data
    if draw:
        plt.plot(x, y, 'x')
        plt.show()
    return x,y
```

Gather some data by calling the function and storing `x` and `y`

Fitting models to data



TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

How to fit a model to the data?

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

$$\hat{y} = a_0 x^3 + a_1 x^2 + a_2 x + a_3$$

How to fit a model to the data?

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

$$\hat{y} = a_0x^3 + a_1x^2 + a_2x + a_3$$

First attempt - using the `polyfit` function we have seen with the interpolation lecture:

```
def fit_using_polyfit(x,y,n=2,draw=False):
    p = np.polyfit(x,y,n)
    if draw:
        plt.plot(x, y, 'x')
        plt.plot(x, np.polyval(p,x), '-')
        plt.show()
    return p
```

If we print `p`, we get the coefficients. But this is a black box, what does it do?

How to fit a model to the data?

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

$$\hat{y} = a_0 x^3 + a_1 x^2 + a_2 x + a_3$$

How to fit a model to the data?

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

$$\hat{y} = a_0 x^3 + a_1 x^2 + a_2 x + a_3$$

Attempt to solve a linear system: If we have N data points, we could write the model as the product of a matrix and a vector:

$$\begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{N-1} \end{bmatrix} = \begin{bmatrix} x_0^3 & x_0^2 & x_0 & 1 \\ x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{N-1}^3 & x_{N-1}^2 & x_{N-1} & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$\hat{y} = Xa$$

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:

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

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 Python, we can solve our linear system $\hat{Y} = Xa$ simply by running `a = np.linalg.solve(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:

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

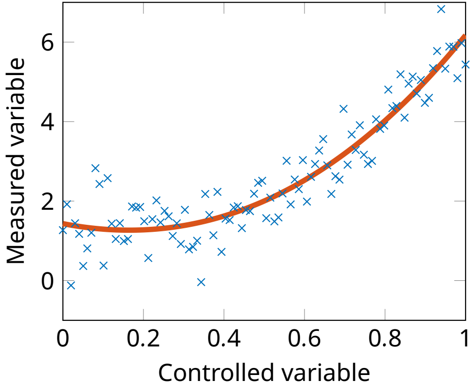
In Python, we can solve our linear system $\hat{Y} = Xa$ simply by running `a = np.linalg.solve(X,y)`.

- 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, Python will find values for the vector a which minimize $\left\|y - aX\right\|^2$, so i.e. a least squares fit.

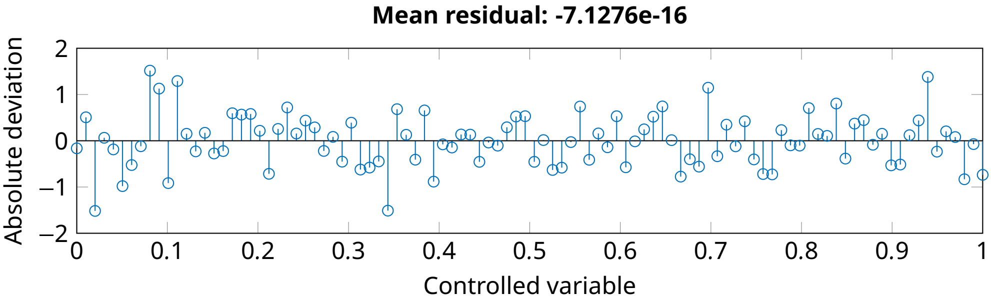
Fitting our problem: Least squares solver

As a follow-up of the script provided in slide 505

```
def fit_using_lstsqr(x,y,n=2,draw=False):
    xmat = np.vander(x,n+1)
    sol = np.linalg.lstsq(xmat,y,rcond=None)
    A = sol[0]
    yhat = xmat@A
    if draw:
        plt.plot(x,y,'x')
        plt.plot(x,yhat,'-')
        plt.title('Fit using lstsq')
        plt.show()
    return A,yhat
```



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.

Today's outline

- Introduction
- Curve fitting
- Regression
- Fitting numerical models
- Optimization
- Linear programming
- Summary

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Regression coefficients

- Variance measured in the data (y) is:

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

- Variance of the residuals is:

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

- Variance in the model is:

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

Regression coefficients

Given that the error is uncorrelated we can state that:

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

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

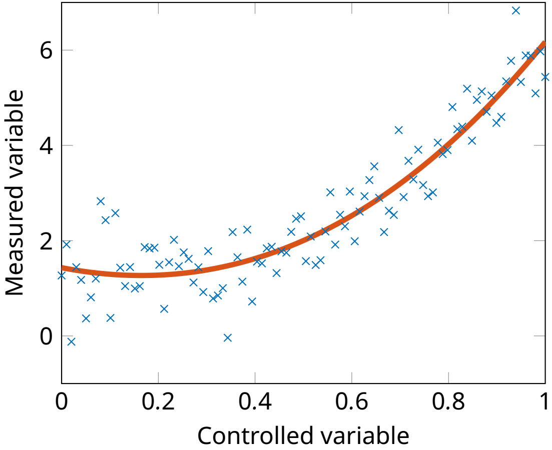
$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

- SSE: Sum of errors (residuals) squared (difference between data and model)
- SST: Total sum of squares (variance of the data)
- SSR: Sum of squares (model)

Back to the example

The statistics:

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



Today's outline

- Introduction
- Curve fitting
- Regression
- Fitting numerical models
- Optimization
- Linear programming
- Summary

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Curve fitting from command line: scipy.optimize.curve_fit

Python offers various non-linear parameter and curve fitting tools that can be run from the command line. The function `curve_fit` from the `scipy.optimize` module allows to fit a model to a given dataset. Again, based on the data generated in slide 505:

```
from poly_regression import generate_random_data
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import numpy as np

# Define the model function
def curve_fit_model_1(xdata, a0, a1, a2, a3):
   return a0*xdata**3 + a1*xdata**2 + a2*xdata + a3
```

```
if __name__ == '__main__':
   x,y = generate_random_data()
   a0 = [1, 2, 1, 3] # Initial guess of coefficients

   # Perform fitting, store resulting coeffs in a_fit
   a_fit, _ = curve_fit(curve_fit_model_1, x, y, p0=a0)

   # Run the model once more, with fitted coefficients
   y_model = curve_fit_model_1(x, *a_fit) # unpack coefficients using *
```

Curve fitting from command line: scipy.optimize.curve_fit

For fitting a polynomial model, this function works as well as `polyfit`. But it also allows other (much more complex) types of model to be defined:

```
def curve_fit_model_2(xdata, a0, a1, a2):
    return a0*np.exp(a1*xdata) + a2

# Initial guess of coefficients
a0 = [1,1,1]

# Perform fitting of the data to another model
a_fit, _ = curve_fit(curve_fit_model_2, x, y, p0=a0)

# Run the model once more, with fitted coefficients
y_model = curve_fit_model_2(x, *a_fit)
```

The model functions take individual parameters separately, we use the unpacking syntax (`*`) to pass the fitted parameters when generating the `y_model` data.

Dynamic fitting of non-linear equations

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. Python's Scipy library gives access to powerful functions such as `least_squares` and `curve_fit`.

General use of scipy.optimize.least_squares

```python
from scipy.optimize import least_squares

result = least_squares(fun, k0, bounds=(lb, ub), xtol=1.0E-6, max_nfev=1000)
```

- `fun` is a function handle to the fit criterion (e.g., `myFitCrit`). The fit criterion 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.
- Use arguments such as `xtol` and `max_nfev` for more fine-grained control on the fit procedure.

General use of scipy.optimize.curve_fit

```python
from scipy.optimize import curve_fit

popt, pcov = curve_fit(fun, xdata, ydata, p0=k0, bounds=(lb, ub))
```

- `fun` is the model function that you want to fit to your data. It takes the independent variable as the first argument and the parameters to fit as separate remaining arguments.
- `xdata` and `ydata` are the data points that you are fitting the model function to.
- `k0` is the initial guess for the parameters to be fitted.
- `lb` and `ub` are the lower and upper bounds for the parameters, respectively.
- `popt` will contain the optimized parameters, and `pcov` will contain the covariance matrix, which can give you an idea of the uncertainties of the estimates.

Example use of scipy.optimize.curve_fit

We have experimental data stored in a file, possibly in a .csv or .txt format, 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 scipy.optimize.curve_fit

We have experimental data stored in a file, possibly in a .csv or .txt format, 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 define a function that describes our model:

```python
import scipy as sp
import numpy as np

def simpleode(t, u, k1, k2):
    dudt = -k1*u + k2
    return dudt
```

Note that we supply the coefficients k_1 and k_2 as arguments to the function.

Example use of scipy.optimize.curve_fit

We have experimental data stored in a file, possibly in a .csv or .txt format, 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 define a function that describes our model:

```
import scipy as sp
import numpy as np

def simpleode(t, u, k1, k2):
    dudt = -k1*u + k2
    return dudt
```

  Note that we supply the coefficients k_1 and k_2 as arguments to the function.
- We create a fit criterion function:

```
def fitcrit(t,k1, k2):
    u0 = [1.0]
    tspan = [0,max(t)]
    sol = sp.integrate.solve_ivp(simpleode, tspan, u0, args=(k1, k2), t_eval=t)
    return sol.y[0]
```

Example use of scipy.optimize.curve_fit

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

Example use of scipy.optimize.curve_fit

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

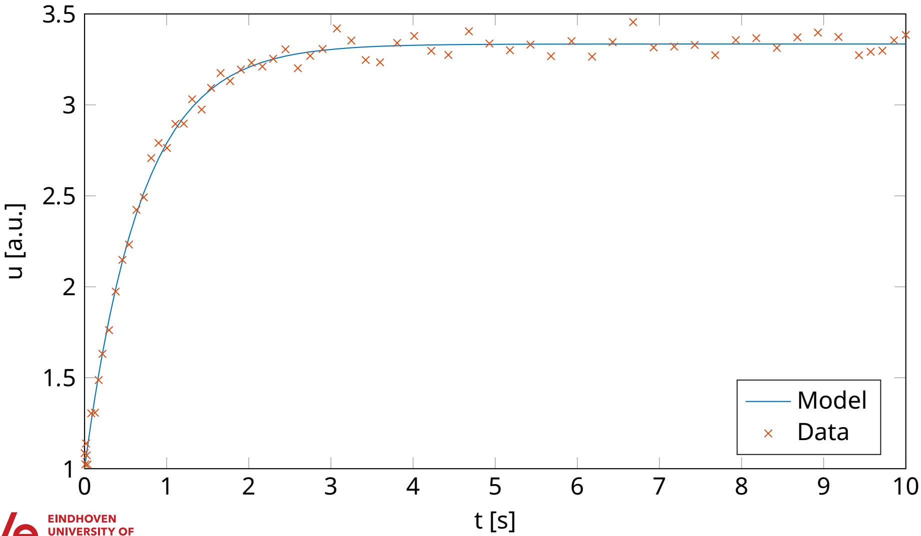
```python
# Load your data here (adjust as necessary)
T, U = np.loadtxt('./scripts/optimization/tudataset1.txt',unpack=True, skiprows=1)

# Initial guesses for model parameters
k0 = [1.0, 1.0]

# Perform the curve fitting
params, params_covariance = sp.optimize.curve_fit(fitcrit, T, U, p0=k0)
print('Fitted coefficients:', params)
```

Our fitted coefficients are stored in `params`. The `params_covariance` gives an estimate of the covariance of the estimated parameters, offering an insight into the uncertainty of the fit.

Example use of lsqnonlin



Postprocessing of results

The data returned by `curve_fit` can be used to obtain the 95% confidence intervals for the fitted parameters. Recall the command:

```
params, params_covariance = curve_fit(fitcrit, T, U, p0=k0)
```

We can use the square root of the diagonal of the covariance matrix, multiplied by a factor from the t-distribution to get the confidence bounds:

```
from scipy import stats
import numpy as np

alpha = 0.05 # 95% confidence interval = 100*(1-alpha)
n = len(U) # number of data points
p = len(params) # number of parameters

dof = max(0, n - p) # number of degrees of freedom
# t value for the dof and confidence level
tval = stats.t.ppf(1.0-alpha/2., dof)
sigma = np.sqrt(np.diag(params_covariance))
ci = sigma * tval

print('Confidence intervals:')
print('k1:', params[0] - ci[0], params[0] + ci[0])
print('k2:', params[1] - ci[1], params[1] + ci[1])
```

Today's outline

- Introduction
- Curve fitting
- Regression
- Fitting numerical models
- Optimization
- Linear programming
- Summary

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 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

Today's outline

- Introduction
- Curve fitting
- Regression
- Fitting numerical models
- Optimization
- Linear programming
- Summary

EINDHOVEN UNIVERSITY OF TECHNOLOGY

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.

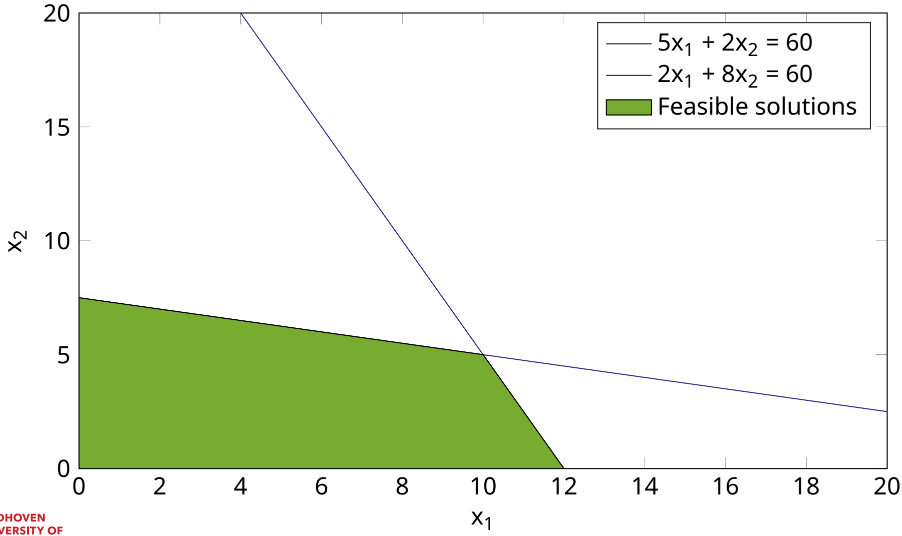
For example:

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

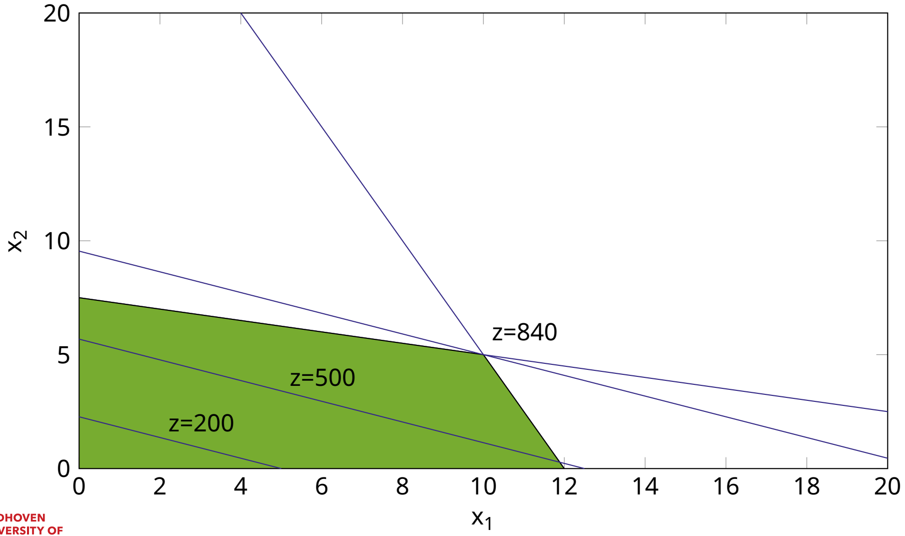
If the constraints are satisfied, but the objective function is not maximized/minimized we speak of a feasible solution.

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

$$\max z = f(x_1, x_2) = 40x_1 + 88x_2$$

s.t.

$$2x_1 + 8x_2 \leq 60$$
$$5x_1 + 2x_2 \leq 60$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

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 Python 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` from `scipy.optimize`:

```python
from scipy.optimize import linprog

c = [-5, -4, -6]
A = [[1, -1, 1], [3, 2, 4], [3, 2, 0]]
b = [20, 42, 30]
bounds = [(0, None), (0, None), (0, None)]

res = linprog(c, A_ub=A, b_ub=b, bounds=bounds)
```

Gives (after accessing appropriate attributes of the result object):

```python
x = res.x
fun = res.fun
slack = res.slack
success = res.success
```

Summary

- Curve fitting: Manual procedures for polynomial fitting in Python
- Curve fitting: Python's SciPy library for curve fitting
- Curve fitting: Python's non-linear least-squares solver `least_squares`
- Curve fitting: Python's non-linear least-squares solver `curve_fit`
- Optimization: An introduction to the Simplex method in Python
- Optimization: Use of the `linprog` function in the SciPy library