UCSC Silicon Valley Extension

Advanced C Programming

Curve Fitting

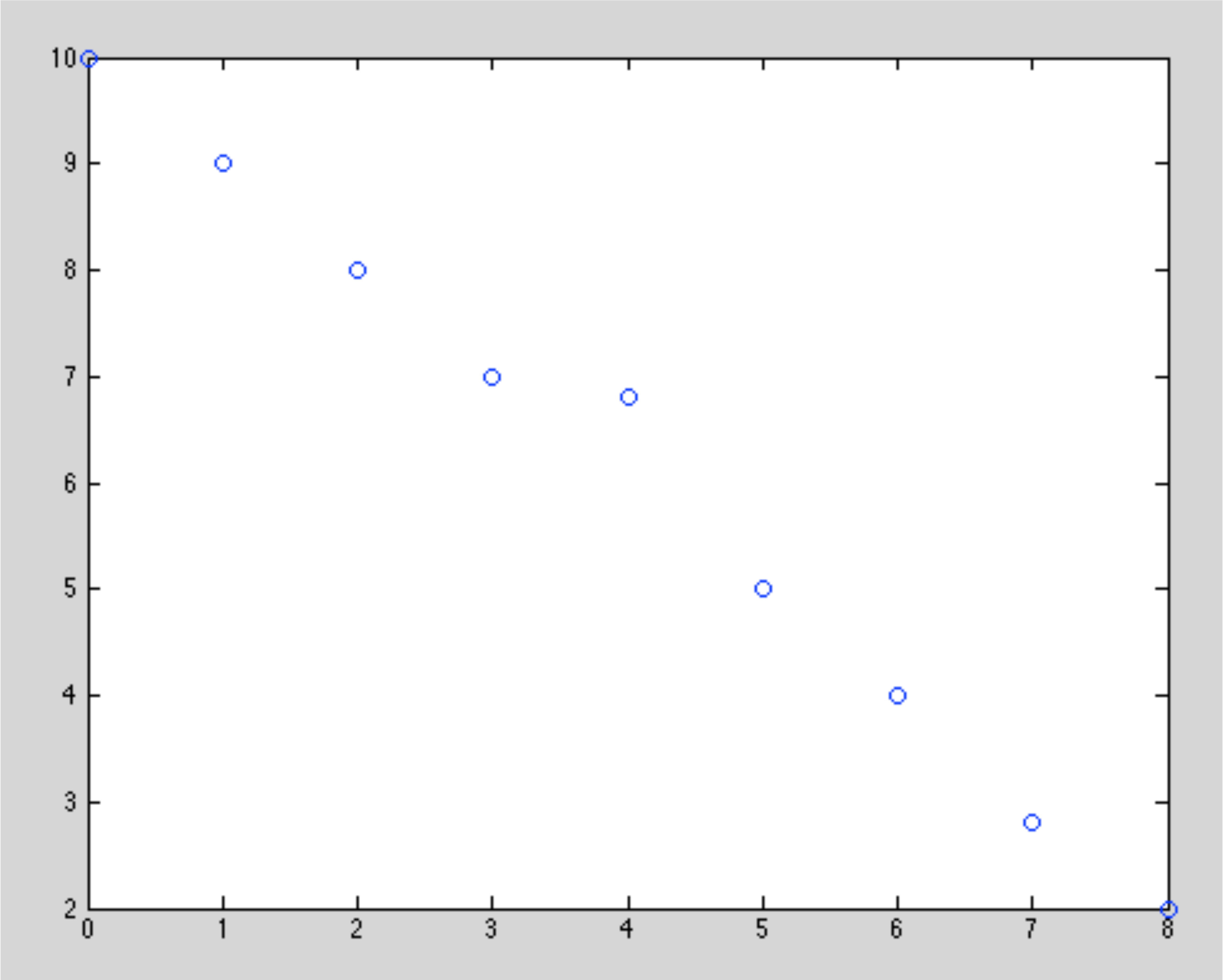
Instructor: Radhika Grover

Curve fitting

- Estimate an equation of the data on the graph.
- Some points will not not fall on the estimated line – these points are called *outliers*.
- We will use GNU Octave, an open source tool for scientific computation.

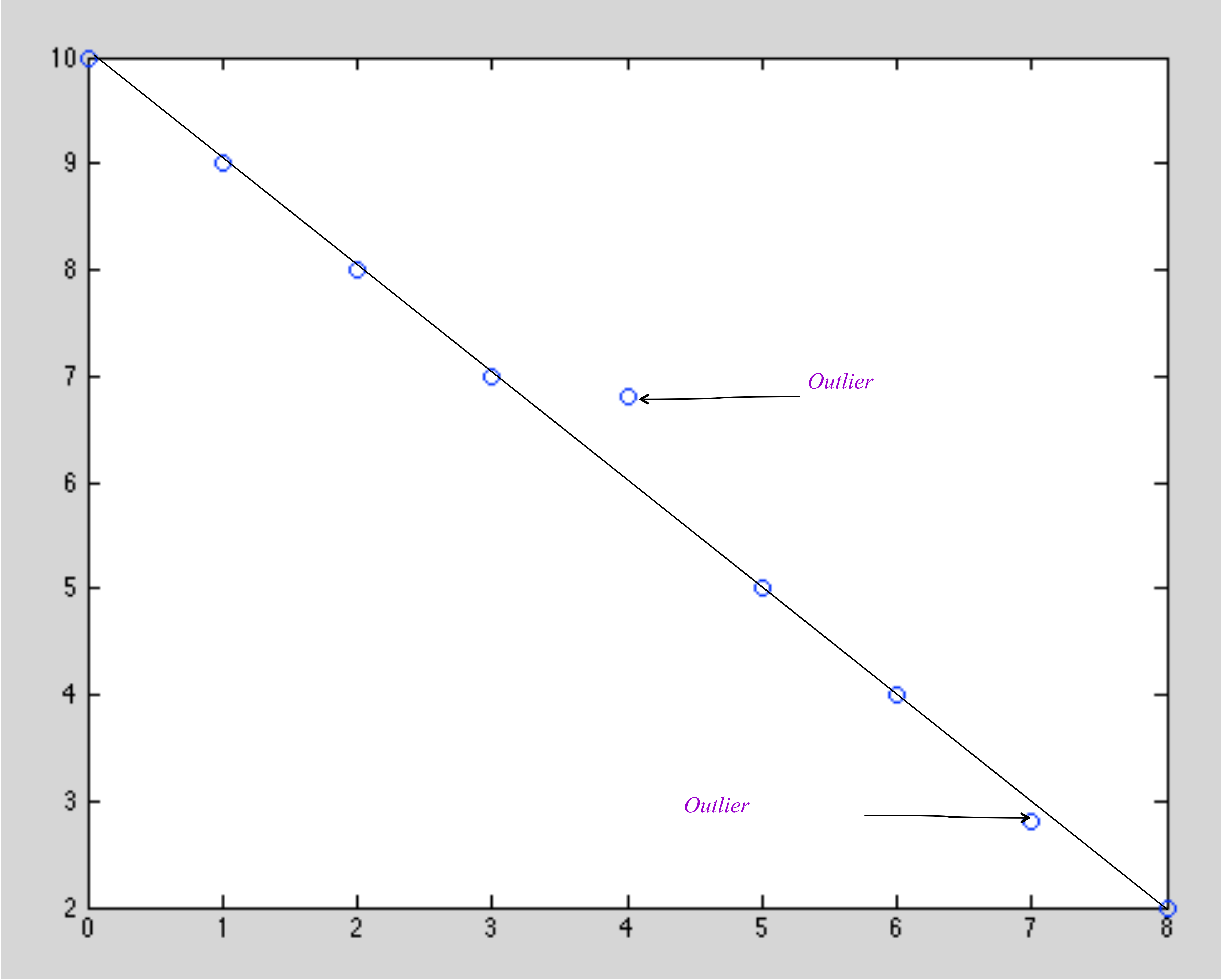
Curve fitting

Find an equation to approximate these data points.



Curve fitting

An approximation of the data points using a straight line:



Curve fitting methods

- **Polynomial regression** – fits the data with a higher-order polynomial.

- Methods:

  p = **polyfit**(*x*, *y*, *n*) % n = 1 for linear; n > 1 for polynomial

  Return the coefficients of a polynomial *p*(*x*) of degree *n* that minimizes the least-squares-error of the fit to the points `[x, y]`.

  polyval(coeff, x)

Linear regression

- Uses method of **least squares**:
  - Compares different equations to model set of data.
  - Differences between actual and calculated values are squared and added together (called ***sum of squares***).
  - Selects equation with the *smallest* sum of squares.

Fitting to a polynomial

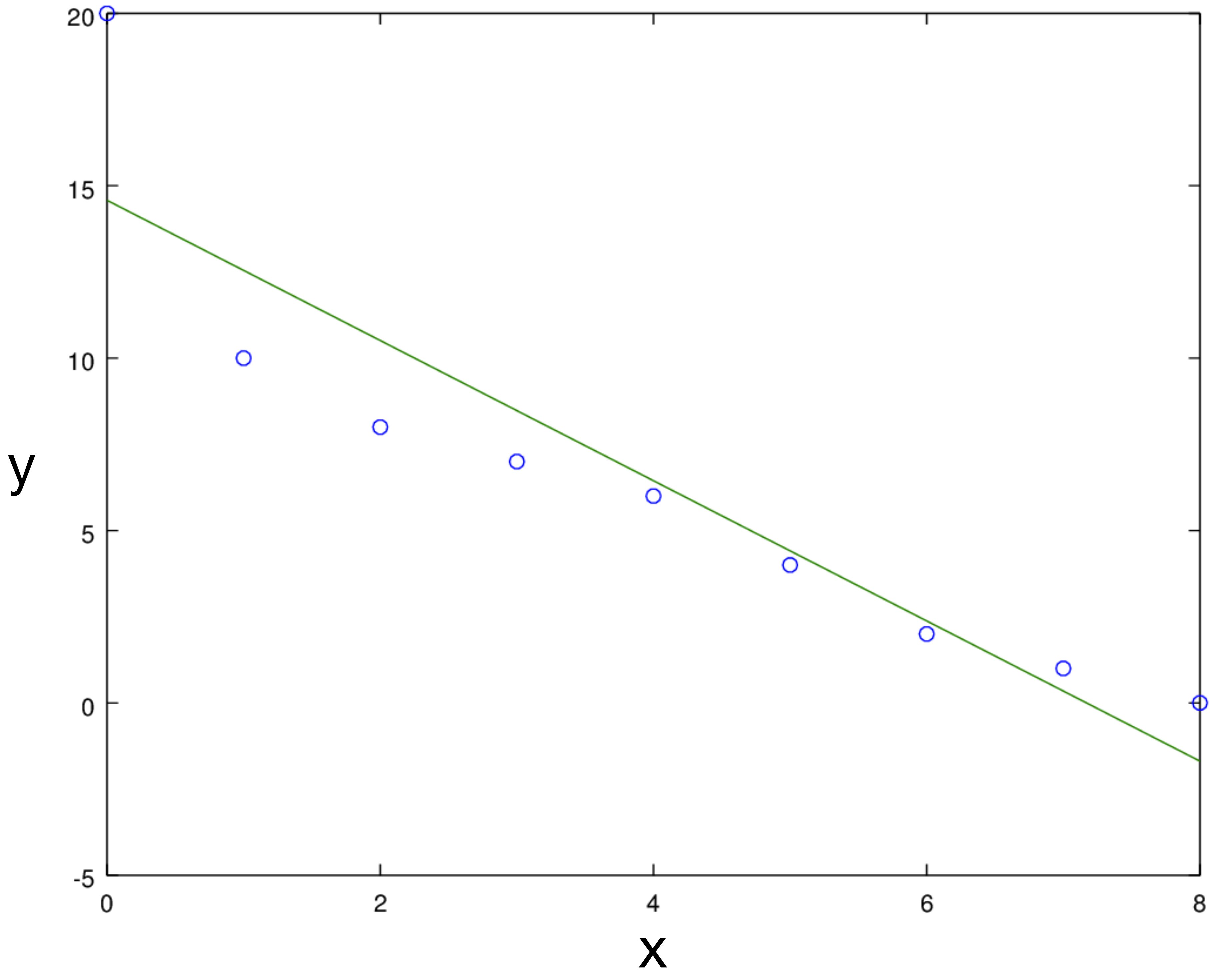
- **polyfit** takes *arrays* x and y containing data points as input and returns *coefficients* that best fit the data.
- **polyval** takes the *coefficients* and *array* x as argument and returns new *array* of y values.

Linear regression: polyfit

- Fits data in x and y to line:

```
>> x = 0:8;

>> y = [20 10 8 7 6 4 2 1 0];

>> coef = polyfit(x, y, 1)

coef =


   -2.0333   14.5778


>> newy = coef(1) *  x + coef(2);

>> plot(x, y, 'o', x, newy)
```

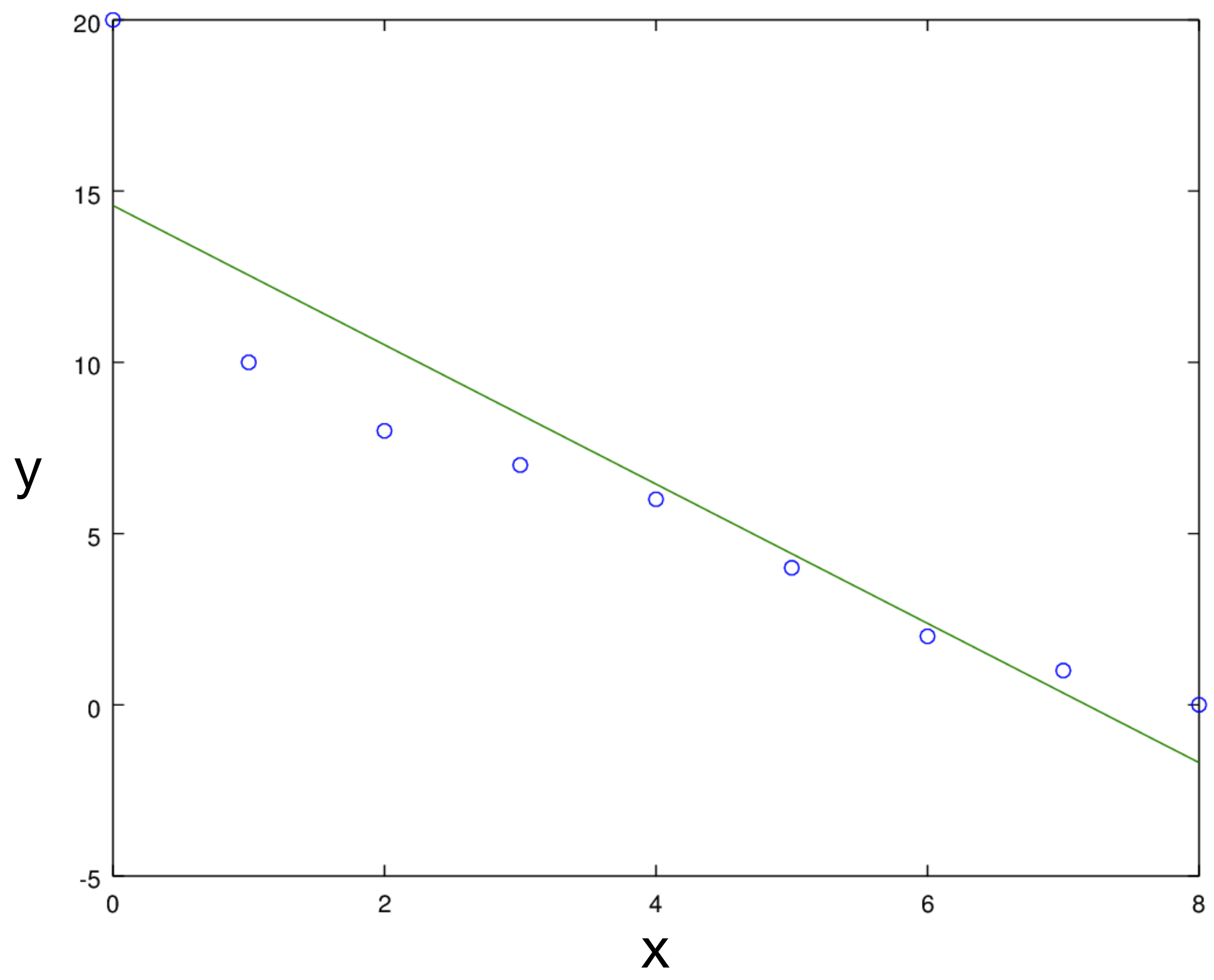
Equation of line that best fits points on graph



Linear regression: polyfit and polyval

- Another way to fit data to line:

```
>> x = 0:8;
>> y = [20 10 8 7 6 4 2 1 0];
>> coef = polyfit(x, y, 1)
coef =

   -2.0333   14.5778

>> newy = polyval(coef, x)
>> plot(x, y, 'o', x, newy)
```



Polynomial regression

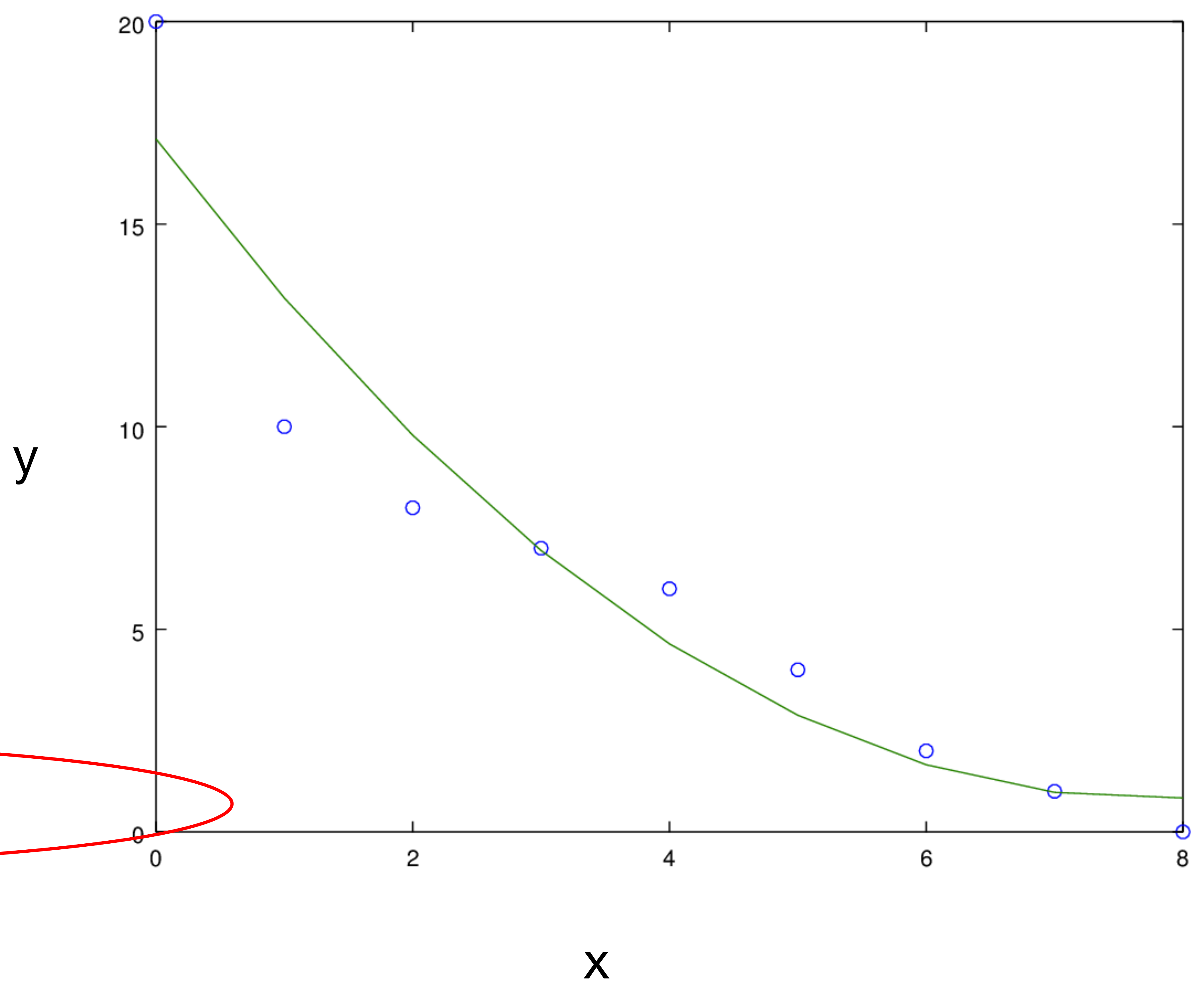
- Don’t use linear regression if the data does not represent a straight line.
- Use polynomial regression instead with $n > 1$
- Finds the coefficients to the following polynomial that fit data best:

$$y = a_1x^n + a_2x^{n-1} + \ldots + a_nx + a_{n+1}$$

Polynomial regression: polyval

- Fits data in x and y to curve of order 2:

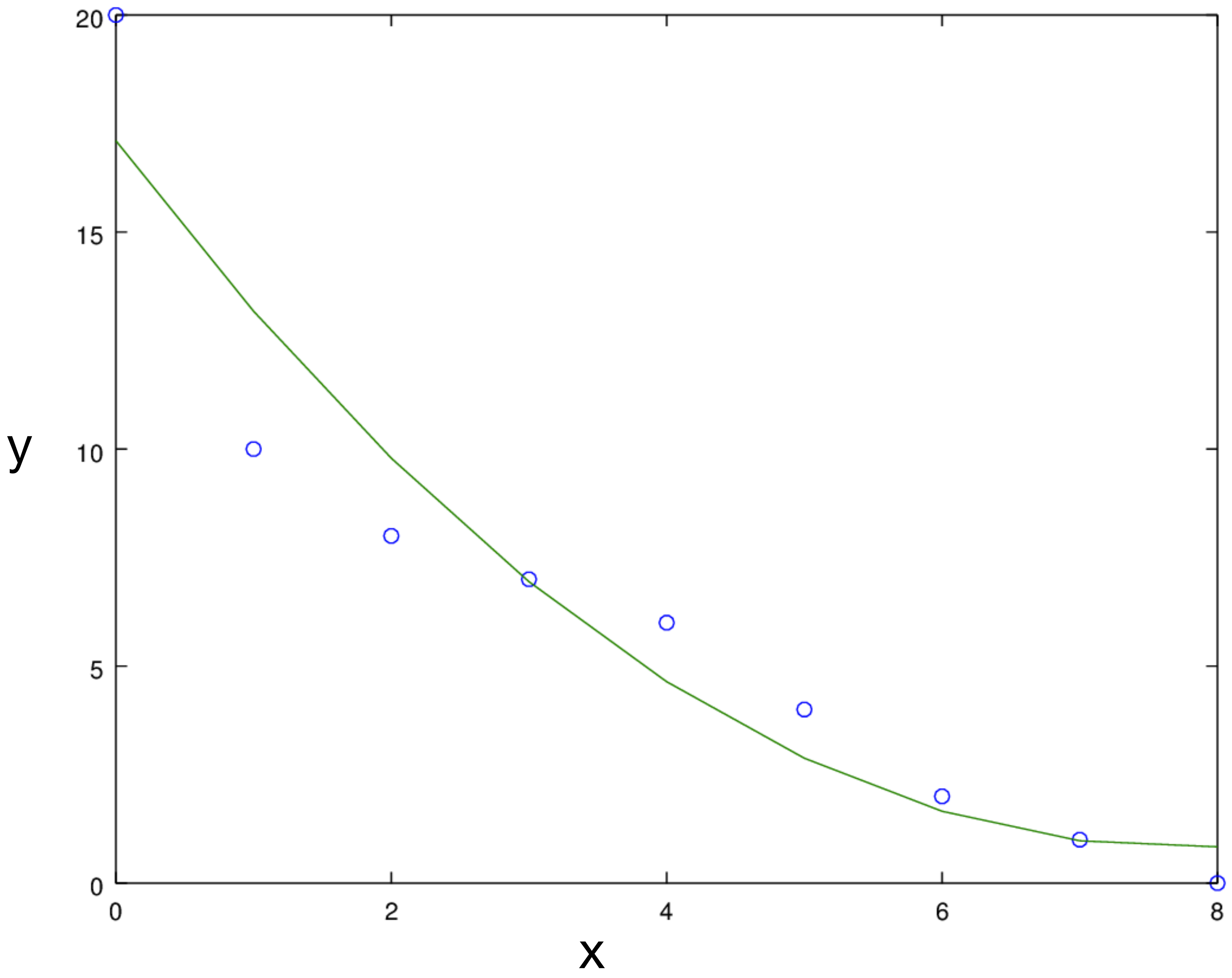
```
>> x = 0:8;
>> y = [20 10 8 7 6 4 2 1 0];
>> coef = polyfit(x, y, 2)
coef =

   0.27056   -4.19784   17.10303

>> ynew = coef(1) * x.^2 + coef(2) * x + coef(3)
>> plot(x, y, 'o', x, ynew)
```



Polynomial regression: polyval and poly fit

- Fits data in x and y to curve of order 2:

```
>> x = 0:8;
>> y = [20 10 8 7 6 4 2 1 0];
>> coef = polyfit(x, y, 2)
coef =

   0.27056   -4.19784   17.10303

>> y2 = polyval(coef, x);
>> plot(x, y, 'o', x, y2)
```



Computing error

- Difference between actual data y_i and fitted data in function $f(x_i)$

- Calculated as root mean square error

$$= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}$$

Least squares fitting :

  - Choosing the parameter of function *f* to minimize the root mean square error .

Example: computing error

y contains actual data

y2 contains fitted data,

```
>> variance = sum((y-y2).^2)/ length(y)

variance =  0.61250

>> sum((y-y2).^2)

ans =  2.4500

>> error = sqrt(2.45)

error =  1.5652
```


Piecewise curve fitting

- Use several polynomials when a single polynomial is not not good.

- `splinefit` fits a piecewise polynomial (spline) to data.

  splinefit *(x, y, breaks)*

Test program

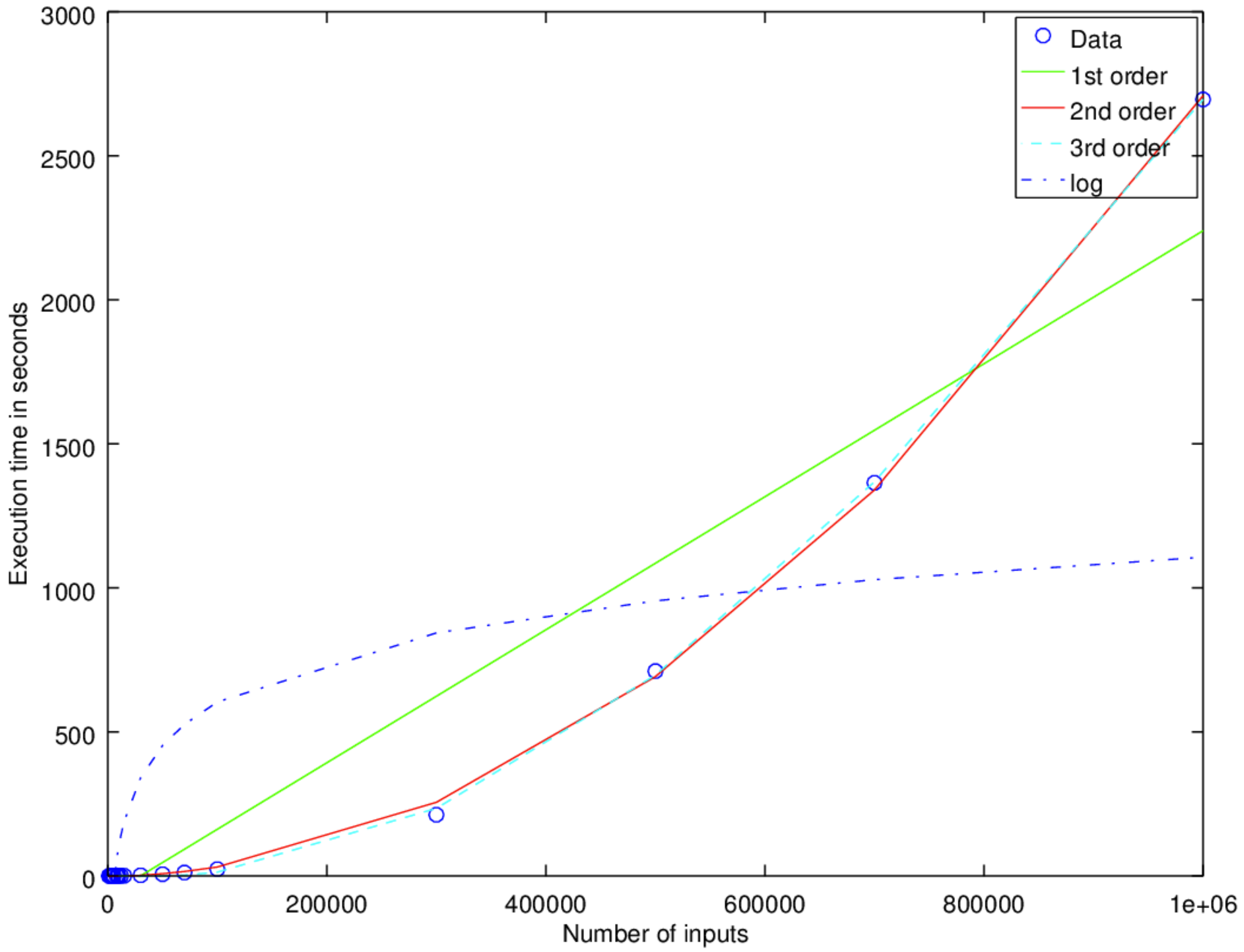
```
#include <iostream>
using namespace std;
#include <time.h>

#define NUM_ITERS 300000

int main() {
   clock_t startTime, endTime;

   startTime = clock();

   for (int i = 0; i <  NUM_ITERS; i++) {
      for(int j = 0; j < NUM_ITERS; j++) {
          int num = num^2 * j ;
      }
   }

   endTime = clock();
   double cpuTime = ((double)(endTime- startTime))/CLOCKS_PER_SEC;

   cout << "hello world" << endl;
   cout << cpuTime << endl;
   return 0;
}
```

Curve fitting the test program

```
x = [1000 2000 2500 3000 3500 4000 4500 5000 5500 8000 9000 10000 12000 15000 30000 50000 70000 100000 300000
500000 700000 1000000];
y = [.003677 0.012282  .016886 0.023896 0.033294  0.041877 0.057448 .069376 0.086146 .192842 .214143 .287536 .400745
.624917 2.1384 5.8697 11.529  23.5158 212.22 710.863 1364.67 2695.57];
plot(x,y,'o')
y1 = polyfit(x, y, 1);   # straight line
coef1 = polyfit(x, y, 1);
y1 = polyval(coef1, x);
hold on;
plot(x, y1, 'g')
coef2 = polyfit(x, y, 2);  # 2nd order polynomial
y2 = polyval(coef2, x);
plot(x, y2, 'r-')
coef3 = polyfit(x, y, 3);  # 3rd order polynomial
y3 = polyval(coef3, x);
plot(x, y3, 'c--')
coef_log = polyfit(log10(x), y, 1);  # log curve
y_log = polyval(coef_log, log10(x))
xlabel('Number of inputs');
ylabel('Execution time in seconds');
xlim([0 1000000]);
ylim([0 3000]);
plot(x, y_log, 'b-.')
legend("Data", "1st order", "2nd order", "3rd order", "log")
```

Array x contains the input size (NUM_ITERS)
Array y contains the execution time (cpuTime)

Plots for test program



Add more data points

References

- https://www.gnu.org/software/octave/
- https://www.gnu.org/software/octave/doc/
- Jaan Klus. Numerical methods in engineering with MATLAB ®. Cambridge University Press, 2009.

