Lesson 13: Data fitting, interpolation

- New concepts:
  - Data fitting as a linear algebra problem
  - Linear regression
  - Normal system solution to least squares problem
  - Interpolation as linear algebra problem

- In many scientific fields the problem of determining the straight line $y(x) = a_0 + a_1 x$, that best approximate data $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, m\}$ arises. The problem is to find the coefficients a_0, a_1, and this is referred to as the linear regression problem.
- The calculus approach: Form sum of squared differences between $y(x_i)$ and y_i

$$S(a_0, a_1) = \sum_{i=1}^{m} (y(x_i) - y_i)^2 = \sum_{i=1}^{m} (a_0 + a_1 x_i - y_i)^2$$

and seek (a_0, a_1) that minimize $S(a_0, a_1)$ by solving the equations

$$\frac{\partial S}{\partial a_0} = 0 \Rightarrow 2\sum_{i=1}^{m} (a_0 + a_1 x_i - y_i) = 0 \Leftrightarrow m\, a_0 + \left(\sum_{i=1}^{m} x_i\right) a_1 = \sum_{i=1}^{m} y_i$$

$$\frac{\partial S}{\partial a_1} = 0 \Rightarrow 2\sum_{i=1}^{m} (a_0 + a_1 x_i - y_i) x_i = 0 \Leftrightarrow \left(\sum_{i=1}^{m} x_i\right) a_0 + \left(\sum_{i=1}^{m} x_i^2\right) a_1 = \sum_{i=1}^{m} x_i\, y_i$$

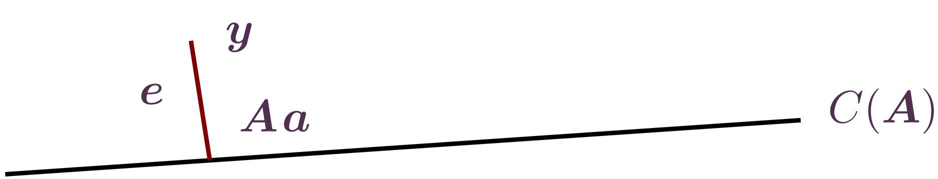
- Form a vector of errors with components $e_i = y(x_i) - x_i$. Recognize that $y(x_i)$ is a linear combination of 1 and x_i with coefficients a_0, a_1, or in vector form

$$\boldsymbol{e} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} - \boldsymbol{y} = \begin{pmatrix} \boldsymbol{1} & \boldsymbol{x} \end{pmatrix} \boldsymbol{a} - \boldsymbol{y} = \boldsymbol{A}\boldsymbol{a} - \boldsymbol{y}$$

- The norm of the error vector $\|\boldsymbol{e}\|$ is smallest when $\boldsymbol{A}\boldsymbol{a}$ is as close as possible to $\boldsymbol{y}$. Since $\boldsymbol{A}\boldsymbol{a}$ is within the column space of $C(\boldsymbol{A})$, $\boldsymbol{A}\boldsymbol{a} \in C(\boldsymbol{A})$, the required condition is for $\boldsymbol{e}$ to be orthogonal to the column space

$$\boldsymbol{e} \perp C(\boldsymbol{A}) \Rightarrow \boldsymbol{A}^T \boldsymbol{e} = \begin{pmatrix} \boldsymbol{1}^T \\ \boldsymbol{x}^T \end{pmatrix} \boldsymbol{e} = \begin{pmatrix} \boldsymbol{1}^T \boldsymbol{e} \\ \boldsymbol{x}^T \boldsymbol{e} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \boldsymbol{0}$$

$$\boldsymbol{A}^T \boldsymbol{e} = \boldsymbol{0} \Leftrightarrow \boldsymbol{A}^T (\boldsymbol{A}\boldsymbol{a} - \boldsymbol{y}) = 0 \Leftrightarrow (\boldsymbol{A}^T \boldsymbol{A}) \boldsymbol{a} = \boldsymbol{A}^T \boldsymbol{y}$$



Linear regression example: Solve $(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{a}=\boldsymbol{A}^T\boldsymbol{y} \Leftrightarrow \boldsymbol{N}\boldsymbol{a}=\boldsymbol{b}$, to find best linear fit

1. Generate some data on a line and perturb it by some random quantities

```
octave> m=1000; x=(0:m-1)/m; a0=2; a1=3; yex=a0+a1*x; y=(yex+rand(1,m)-0.5)';
octave>
```

2. Form the matrices $\boldsymbol{A}$, $\boldsymbol{N}=\boldsymbol{A}^T\boldsymbol{A}$, vector $\boldsymbol{b}=\boldsymbol{A}^T\boldsymbol{y}$

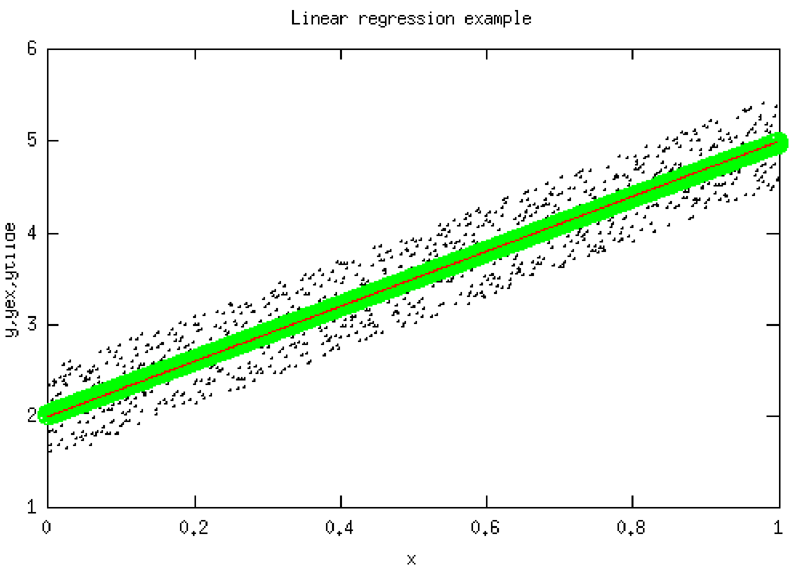
```
octave> A=ones(m,2); A(:,2)=x(:); N=A'*A; b=A'*y;
octave>
```

3. Solve the system $\boldsymbol{N}\boldsymbol{a}=\boldsymbol{b}$, and form the linear combination $\tilde{\boldsymbol{y}}=\boldsymbol{A}\boldsymbol{a}$ closest to $\boldsymbol{y}$

```
octave> a=N\b; disp(a'); ytilde=A*a;
   2.0182   2.9738
octave>
```

- Plot the perturbed data (black dots), the result of the linear regression (green circles), as well as the line used to generate yex (red line)

```
octave> plot(x,y,'.k',x,ytilde,'og',x,yex,'r'); title('Linear regression example');
        xlabel('x'); ylabel('y,yex,ytilde');  cd /home/student; print data.eps;
```



- The key observation is that the matrix $\boldsymbol{A}$ has columns obtained by evaluating the functions $1, x$ at the values $x_1, x_2, \ldots, x_m$. This leads to easy extension to data fitting to higher degree polynomials, for instance a quadratic

$$\boldsymbol{e} = \left(\begin{array}{ccc} \boldsymbol{1} & \boldsymbol{x} & \boldsymbol{x}^2 \end{array} \right) \boldsymbol{a} - \boldsymbol{y} = \boldsymbol{A}\boldsymbol{a} - \boldsymbol{y}, \min \|\boldsymbol{e}\| \Rightarrow (\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{a} = \boldsymbol{A}^T\boldsymbol{y} \Leftrightarrow \boldsymbol{N}\boldsymbol{a} = \boldsymbol{b}$$

```
octave> m=1000; x=(0:m-1)/m; a0=2; a1=3; a2=-5.; yex=a0+a1*x+a2*x.^2; y=(yex+rand(1,m)-
        0.5)';
octave> A=ones(m,3); A(:,2)=x(:); A(:,3)=x.^2; N=A'*A; b=A'*y;
octave> a=N\b;
octave> ytilde=A*a; disp(a');
```

```
2.0239   2.8873  -4.8881
```

```
octave> disp(norm(y-ytilde)/norm(y)/m);
```

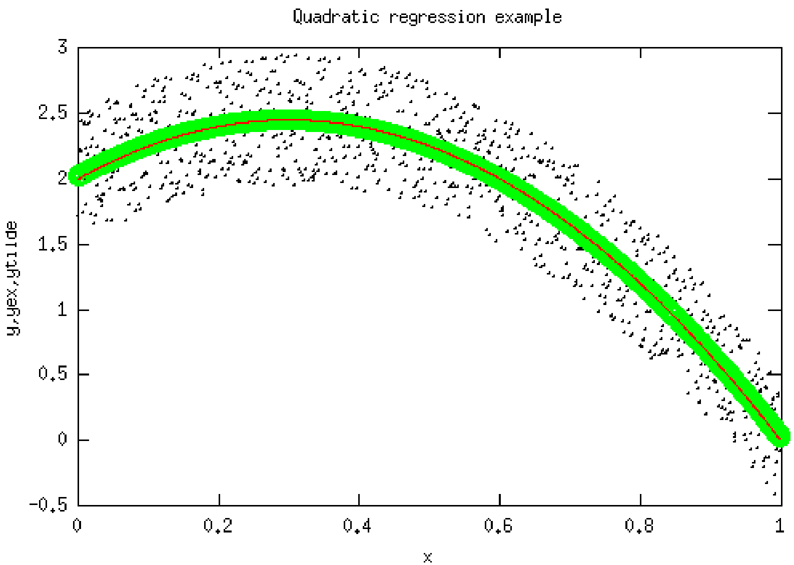
```
1.4494e-04
```

```
octave>
```

Quadratic regression result

- Plot the perturbed data (black dots), the result of the quadratic regression (green circles), as well as the parabola used to generate yex (red line)

```
octave> plot(x,y,'.k',x,ytilde,'og',x,yex,'r'); title('Quadratic regression example');
        xlabel('x'); ylabel('y,yex,ytilde'); cd /home/student; print data.eps;
octave>
```



- Up to now we have considered linear data fitting
- When the data conforms to a non-linear law,it is often possible to transform the problem into a linear dependency
- Example: Find best fit of coefficients A, E_a within Arrhenius law $k = A\exp(-E_a/(RT))$ to measured data $\mathcal{D} = \{(T_i, k_i), i = 1, \ldots, m\}$.
- Note that in the $k(T)$ law, k depends linearly on A, but nonlinearly on E_a. By taking the natural logarithm, and setting $y = \ln k$, $x = 1/(RT)$, $a_0 = \ln A$, $a_1 = -E_a$, we obtain a linear dependence

$$y = \ln k = \ln A - E_a x = a_0 + a_1 x$$

of the same type as before

Definition. *The polynomial interpolant of data* $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, m\}$ *with* $x_i \neq x_j$ *if* $i \neq j$ *is a polynomial of degree* $m-1$

$$p_{m-1}(x) = a_0 + a_1 x + \ldots + a_{m-1} x^{m-1}$$

that satisfies the conditions $p_{m-1}(x_i) = y_i$, $i = 1, \ldots, m$.

- We can apply the same approach, and formulate the normal equation system. In this particular case, the error e can be made zero.

```
octave> m=4; x=(0:m-1)'; a0=2; a1=3; a2=-5.; a3=-1; yex=a0+a1*x+a2*x.^2+a3*x.^3;
```

```
octave> A=ones(m,m); A(:,2)=x(:); A(:,3)=x.^2; A(:,4)=x.^3; N=A'*A; b=A'*yex;
octave> a=N\b; disp(a');
```

```
   2.00000   3.00000  -5.00000  -1.00000
```

Note that the coefficients used to generate the data are recovered exactly.