NMST539

Mnohorozměrná analýza
Multivariate Analysis
(Data Science 3)

Lecture Notes O (prerequisites)

Ivan Mizera

Matrix Trix

Arrays of numbers; matrices

An **array** of numbers is a (rectangular) structure with its **elements** (or **components**) indexed by several integers $\mathrm{i_k}$ in the range $1, 2, \ldots, \mathrm{n_k}$; we speak about $\mathrm{n_1} \times \mathrm{n_2} \times \cdots \times \mathrm{n_d}$ array

The most prominent instance of such an array are matrices, with $\mathrm{d} = 2$; they are represented as $\mathrm{p} \times \mathrm{q}$ tables with elements $\mathrm{a_{ij}}$, where customarily i indexes a row and j a column of such a table

There is some confusion with words "size" and "dimension" here: it may happen that one speaks about a matrix with dimension $\mathrm{p} \times \mathrm{q}$, but in the context of arrays, every matrix has **dimension** $\mathrm{d} = 2$, and then may have **size** $\mathrm{p} \times \mathrm{q}$

If one or several of the indices are kept fixed, the pertinent elements form subarrays; for matrices, the important subarrays are **rows** (row index is fixed to one value) and **columns** (column index is fixed to one value)

Some history

Calculus of matrices turned out to be a very useful tool in quantum mechanics theory - around 1920's - and then rapidly spread to other parts of mathematics and computing. Matrix algebra allows for compact expressions, and is often a sufficient tool in certain areas of applications

Before matrices became popular, it was habitual to work with arrays, even more general than matrices, in a componentwise notation - the algebra of *tensors*. As tensors have specific meaning in physics, we rather stick to a word "array" here - similarly as the R software environment does. The advantage of the componentwise notation is its generality; its disadvantage is a certain obscurity of the formulas (which may be quite in the eyes of the beholder)

Vectors

Matrix algebra became so popular that other arrays are often converted to matrices to facilitate manipulations with them (and thus avoid componentwise notation). The easiest to convert are

vectors - the arrays with only one index, with components $a_1, a_2, \ldots, a_p$. They can be accommodated into matrix algebra either as $1 \times p$ or $p \times 1$ matrices - the latter being the convention we will hereafter adhere to

(Interestingly, the software environment R supports "ambiguity" in the representation of vectors: they may be either lines or columns, depending on the context)

Some advanced part of statistics (nonlinear regression or multivariate analysis, for instance) require from time to time working with arrays other than matrices or vectors. To avoid the componentwise notation here, various ways to accommodate arrays into matrix algebra were devised. The simplest one is *vectorization*: the d-dimensional array is converted into a vector in a reverse lexicographic order: the last index changes least often. For matrices it means that columns are stacked into one column in their order - a *block vector* is made out of columns

Block matrices, vectors, and other tidbits

We can form arrays from smaller arrays by stacking them in one index, provided that other indices of the stacked arrays are matched. A common way is to form a bigger matrix from submatrices, or a vector by stacking several vectors. A matrix can be considered stacked from columns or rows.

Matrix algebra of block matrices and vectors is guided by a simple principle: blocks behave analogously as simple elements, provided that *all the dimensions match* - regarding stacking and multiplication

The transposition of matrices is denoted by $^{\mathsf{T}}$, symbol $'$ being reserved for derivatives

The determinant of a square matrix is written as $\det(\mathbf{A})$ - the notation that allows for considering its absolute value as $|\det(\mathbf{A})|$

A **diagonal matrix** is any one that has nonzero elements possible only on the diagonal: $\mathbf{a_{ij}} = 0$ for $\mathbf{i} \neq \mathbf{j}$; it is not necessarily a square matrix. On the other hand, $\mathrm{diag}(\mathbf{u})$ denotes the $\mathbf{p} \times \mathbf{p}$ diagonal matrix with $\mathbf{u}$ forming its diagonal

Componentwise

Despite the versatility of matrix calculus, in some situations is better to use the componentwise notation; for instance, matrix multiplication of a $\mathrm{p} \times \mathrm{q}$ matrix **A** by a $\mathrm{q} \times \mathrm{r}$ matrix **B** yields elementwise a $\mathrm{p} \times \mathrm{r}$ matrix **C** with elements indexed by i and k

$$\mathrm{c_{ik}} = \sum_{\mathrm{j}} \mathrm{a_{ij}} \mathrm{b_{jk}}$$

One can then also easily handle the componentwise multiplication (a default in R, the `.*` multiplication in MATLAB): if both **A** and **B** are both $\mathrm{p} \times \mathrm{q}$ matrices, then their componentwise product (sometimes called *Hadamard product*) is a $\mathrm{p} \times \mathrm{q}$ matrix **C** with the elements

$$\mathrm{c_{ij}} = \mathrm{a_{ij}} \mathrm{b_{ij}}$$

Such products are sometimes converted to matrix products, but the result may be less intuitive: for instance, when one wants to multiply every line of a $\mathrm{p} \times \mathrm{q}$ matrix **B** (a vector if $\mathrm{q} = 1$) by corresponding elements of a $\mathrm{p} \times 1$ vector $\mathbf{u}$, one has to express a matrix with elements $\mathrm{c_{ij}} = \mathrm{u_i} \mathrm{b_{ij}}$ as $\mathrm{diag}(\mathbf{u})\mathbf{B}$,

Trace: a trivial but useful tool

The trace of a (square) matrix is the sum of its diagonal elements:

$$\mathrm{tr}(A) = \sum_i a_{ii}$$

A useful property of the trace is

$$\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$$

This is a very simple example where the componentwise notation really helps, as

$\sum_j \sum_i a_{ij} b_{ji}$ corresponds to the left-hand side and

$\sum_i \sum_j a_{ji} b_{ij}$ corresponds to the right-hand side

in particular, $\mathrm{tr}(\mathbf{AB}^\mathsf{T}) = \mathrm{tr}(\mathbf{B}^\mathsf{T}\mathbf{A}) = \mathrm{tr}(\mathbf{A}^\mathsf{T}\mathbf{B}) = \mathrm{tr}(\mathbf{BA}^\mathsf{T}) = \sum_{i,j} a_{ij} b_{ij}$

The property is often used in the so-called "trace trick": when one of the product matrices has size 1×1, then its single element is is equal to its trace

Norms, distances, angles

The **Euclidean norm** of any array is the square root of the sum of squares of all its elements; the norm of a vector $\mathbf{x}$ is $\|\mathbf{x}\| = \sqrt{\mathbf{x}^\mathsf{T}\mathbf{x}}$

For matrices, the Euclidean norm is often called *Hilbert-Schmidt* or *Frobenius norm*; the norm of a general matrix $\mathbf{A}$ is equal to

$$\sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\mathrm{tr}(\mathbf{A}^\mathsf{T}\mathbf{A})} = \sqrt{\mathrm{tr}(\mathbf{A}\mathbf{A}^\mathsf{T})}$$

In the matrix algebra context, no special notation for scalar (inner) product is needed: we can write $\mathbf{x}^\mathsf{T}\mathbf{y}$, which is the same as $\mathbf{y}^\mathsf{T}\mathbf{x}$

The Euclidean distance of two vectors (points) is: $\|\mathbf{x} - \mathbf{y}\|$

The angle between two vectors is given by: $\cos\varphi = \dfrac{\mathbf{x}\mathbf{y}}{\|\mathbf{x}\|\,\|\mathbf{y}\|}$

(Cauchy)-(Schwarz)-((Bunjakovski)) inequality says: $|\mathbf{x}\mathbf{y}| \leqslant \|\mathbf{x}\|\,\|\mathbf{y}\|$ the equality true only when $\mathbf{x}$ and $\mathbf{y}$ are collinear (linearly dependent)

Orthogonal matrices and transformations

Matrices correspond to the transformations they represent - to an extent that we often use the same adjective for matrices and transformations: for instance, we speak about **invertible** matrices/transformations

(Note: the adjective "invertible" for matrices is preferred to the adjective "regular", which in English terminology may have different meaning - although "singular" usually means the same)

A matrix is called **orthogonal**, if its transpose is its inverse:
$\mathbf{U}^\top = \mathbf{U}^{-1}$ or equivalently, $\mathbf{U}^\top\mathbf{U} = \mathbf{U}\mathbf{U}^\top = \mathbf{I}$

(Orthogonal matrices thus have *orthonormal* rows and columns; but they are customarily not called orthonormal themselves)

Orthogonal transformations, transformations represented by orthogonal matrices, preserve Euclidean distances and angles. The determinant of an orthogonal matrix is equal to -1 or 1; an important subclass of orthogonal transformations with determinant equal to 1 are *rotations*; beware that applied data-analytic terminology tends often to identify the class of rotations with the class of all orthogonal transformations

Diagonalization

Many operations on matrices are much simpler when performed on diagonal matrices; thus, change of coördinate systems that makes a matrix diagonal can be quite helpful.

If this is possible for a matrix $\mathbf{A}$, it means that there is an invertible matrix $\mathbf{U}$ such that $\mathbf{A} = \mathbf{ULU}^{-1}$ and $\mathbf{L}$ is a diagonal matrix

For *symmetric* matrices, we enjoy a stronger form of diagonalizability, in which $\mathbf{U}$ can be taken *orthogonal*: every symmetric matrix $\mathbf{A}$ can be written in a form $\mathbf{A} = \mathbf{ULU}^{\mathrm{T}}$, where $\mathbf{U}$ is an orthogonal matrix and $\mathbf{L}$ is a diagonal matrix

Recall: if $\mathbf{Ax} = \lambda\mathbf{x}$ for a nonzero vector $\mathbf{x} \neq \mathbf{0}$, then λ is called an **eigenvalue** of $\mathbf{A}$ and $\mathbf{x}$ is its corresponding **eigenvector** (that is, one of these eigenvectors, as any $\mathbf{cx}$ for $\mathrm{c} \neq 0$ qualifies too)

It is easy to see then that the diagonal of $\mathbf{L}$ consists of all eigenvalues and $\mathbf{U}$ consists of their corresponding eigenvectors with unit norm. A $\mathrm{p} \times \mathrm{p}$ symmetric matrix $\mathbf{A}$ has thus at most p eigenvalues (and all its eigenvectors and eigenvalues are real)

Symmetrization

While the diagonalization tricks can be performed also for $\mathbf{A}$ more general than symmetric, given the fact that we will need that pretty much exclusively for handling quadratic forms, there is no need for complications in that direction; matrices generating quadratic forms can be considered symmetric without loss of generality:

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} = \mathbf{x}^\top \left(\frac{1}{2}(\mathbf{A} + \mathbf{A}^\top) \right) \mathbf{x} \qquad \text{as } (\mathbf{A}\mathbf{B})^\top = \mathbf{B}^\top \mathbf{A}^\top$$

This is important, as the original matrix $\mathbf{A}$ may not be diagonalizable, while its *symmetrization* $(\mathbf{A} + \mathbf{A}^\top)/2$ always is

Nonnegative definite or positive semidefinite?

A symmetric matrix $\mathbf{A}$ is **nonnegative definite** (sometimes they also say *positive semidefinite*) if $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} \geqslant 0$ for every $\mathbf{x}$

The diagonalization trick shows that it is if and only if all the eigenvalues of $\mathbf{A}$ are nonnegative. In such a case, we can form a square root of matrix: $\mathbf{A}^{1/2} = \mathbf{U}\mathbf{L}^{1/2}\mathbf{U}^{\mathsf{T}}$

Matrix $\mathbf{A}$ is **positive definite** if $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} \geqslant 0$ for every $\mathbf{x} \neq \mathbf{0}$. This is true if and only if all its eigenvalues are positive; in that case it is also invertible ("regular"), as that is ture when all the eigenvalues are nonzero - the inverse is then $\mathbf{U}\mathbf{L}^{-1}\mathbf{U}^{\mathsf{T}}$

Diagonalization helps

The above shows how certain operations can be extended to diagonalizable matrices: first you define the operation for diagonal matrices (usually componentwise) and then use diagonalization. For instance, one can define $\mathrm{e}^{\mathbf{A}}$ in this way

The trace of a matrix is a sum of its eigenvalues

The eigenvalues of a symmetric and idempotent ($\mathbf{AA} = \mathbf{A}$) matrix are equal to the squares of themselves: thus they are either 0 or 1. The rank of $\mathbf{A}$ is therefore equal to its trace

Matrices $\mathbf{AB}$ and $\mathbf{BA}$ (when both products are square matrices) have the same *nonzero* eigenvalues

The theorem of Eckart and Young

Let $\mathbf{S}$ be a symmetric nonnegative definite matrix; its best approximation by a symmetric matrix, in the Hilbert-Schmidt norm, that has rank at most $\mathfrak{m}$, is the matrix $\mathbf{U}\mathbf{L}_{\mathfrak{m}}\mathbf{U}^{\mathsf{T}}$, where $\mathbf{S} = \mathbf{U}\mathbf{L}\mathbf{U}^{\mathsf{T}}$ is the eigenvalue decomposition of $\mathbf{S}$, and $\mathbf{L}_{\mathfrak{m}}$ is the matrix formed from $\mathbf{L}$ by retaining the $\mathfrak{m}$ largest eigenvalues, and replacing everything else by zero.

Let $\mathbf{A}$ be an arbitrary matrix. The Hilbert-Schmidt distance of $\mathbf{S}$ and $\mathbf{A}$ is

$$\begin{aligned}
\mathrm{tr}\,((\mathbf{S}-\mathbf{A})(\mathbf{S}-\mathbf{A})^{\mathsf{T}}) &= \mathrm{tr}\,(\mathbf{U}\mathbf{U}^{\mathsf{T}}(\mathbf{S}-\mathbf{A})\mathbf{U}\mathbf{U}^{\mathsf{T}}(\mathbf{S}-\mathbf{A})^{\mathsf{T}}) \\
&= \mathrm{tr}\,(\mathbf{U}^{\mathsf{T}}(\mathbf{S}-\mathbf{A})\mathbf{U}\,\mathbf{U}^{\mathsf{T}}(\mathbf{S}-\mathbf{A})^{\mathsf{T}}\mathbf{U}) \\
&= \mathrm{tr}\,((\mathbf{U}^{\mathsf{T}}\mathbf{S}\mathbf{U}-\mathbf{U}^{\mathsf{T}}\mathbf{A}\mathbf{U})(\mathbf{U}^{\mathsf{T}}\mathbf{S}\mathbf{U}-\mathbf{U}^{\mathsf{T}}\mathbf{A}\mathbf{U})^{\mathsf{T}}) \\
&= \mathrm{tr}\,((\mathbf{L}-\mathbf{U}^{\mathsf{T}}\mathbf{A}\mathbf{U})(\mathbf{L}-\mathbf{U}^{\mathsf{T}}\mathbf{A}\mathbf{U})^{\mathsf{T}}) \text{ etc.}
\end{aligned}$$

Rank

The linear space generated by the columns of an $\mathfrak{n} \times \mathfrak{p}$ matrix $\mathbf{X}$ is $\mathcal{M}(\mathbf{X}) = \mathrm{Im}(\mathbf{X}) = \{\mathbf{X}\mathbf{y} : \mathbf{y} \in \mathbb{R}^{\mathfrak{p}}\}$

Its dimension is equal to the rank of the matrix, $\mathrm{rank}(\mathbf{X})$

Rank of the matrix is equal to the rank of its transpose, which in turn is equal to the rank of their product

$$\mathrm{rank}(\mathbf{X}) = \mathrm{rank}(\mathbf{X}^\top) = \mathrm{rank}(\mathbf{X}^\top\mathbf{X})$$

(Otherwise, the rank of a product of two matrices is only $\leqslant$ than the ranks of each)

Kronecker product

One way of converting the 4-dimensional array with elements that have the special product form $a_{ij}b_{k\ell}$ to a matrix is as follows: if we consider a_{ij} to be the elements of a $p \times q$ matrix **A**, and respectively b_{ij} to be the elements of an $r \times s$ matrix **B**, then their **Kronecker product** (sometimes called *tensor product*) is a $pr \times qs$ matrix

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1q}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2q}\mathbf{B} \\ \vdots & \vdots & & \vdots \\ a_{p1}\mathbf{B} & a_{p2}\mathbf{B} & \dots & a_{pq}\mathbf{B} \end{pmatrix}$$

The eigenvalues of $\mathbf{A} \otimes \mathbf{B}$ are all the products of of the eigenvalues of **A** and eigenvalues of **B**; in the coördinatewise notation,

$$\sum_j a_{ij}u_j = \lambda_u u_i \quad \text{and} \quad \sum_\ell b_{k\ell}v_k = \lambda_v v_k \quad \text{implies}$$

$$\sum_j \sum_\ell a_{ij}b_{k\ell}u_j v_k = \lambda_u \lambda_v u_i v_k$$

Derivatives of functions with matrices

For a function F defined on arrays $\mathbf{X}$ of the same dimension and size, with elements x, we define $\frac{\partial F(\mathbf{X})}{\partial \mathbf{X}}$ to be the array with elements $\frac{\partial F(\mathbf{X})}{\partial x}$ (and the same dimension and size as $\mathbf{X}$). If $\mathbf{X}$ is a matrix, $\frac{\partial F(\mathbf{X})}{\partial \mathbf{X}}$ is the matrix with $\frac{\partial F(\mathbf{X})}{\partial X_{ij}}$ in i-th row and j-th column

We have - mostly by straightforward componentwise verification -

$$\frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \qquad \frac{\partial \mathbf{x}^T\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{a}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{a}^T \qquad \frac{\partial \log\det(\mathbf{X})}{\partial \mathbf{X}} = (\mathbf{X}^{-1})^T$$

Differentiation is sometimes used to verify that something is a solution of a minimization problem - but for the complete verification, one seldom wants to take also second derivatives. A better strategy is to use the Cauchy-Schwarz inequality, or convexity

Singular value decomposition (SVD)

Let $\mathbf{A}$ be an *arbitrary* $\mathrm{p} \times \mathrm{q}$ matrix

Singular value decomposition (SVD): $\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal - $\mathrm{p} \times \mathrm{p}$ and $\mathrm{q} \times \mathrm{q}$, respectively

and $\boldsymbol{\Lambda}$ is $\mathrm{p} \times \mathrm{q}$ diagonal with diagonal elements

$\lambda_{\mathrm{i}} \geqslant 0$ (singular values)

Let us test it in R

```
> A = cbind(c(1,2,3),c(2,5,4))
> A
     [,1] [,2]
[1,]    1    2
[2,]    2    5
[3,]    3    4
```

???

```
> sa=svd(A)
> sa
$d
[1] 7.6203733 0.9643188
$u
           [,1]        [,2]
[1,] -0.2932528 -0.08121183
[2,] -0.7017514 -0.65838502
[3,] -0.6492670  0.74828724
$v
           [,1]       [,2]
[1,] -0.4782649  0.8782156
[2,] -0.8782156 -0.4782649
> sa$u %*% diag(sa$d) %*% t(sa$v)
     [,1] [,2]
[1,]    1    2
[2,]    2    5
[3,]    3    4
```

Economy class

Again, there are two versions of SVD: the one introduced above, and the other, "economy" version, in which:

- if $\mathrm{p} \geqslant \mathrm{q}$: $\mathbf{V}$ as above, $\boldsymbol{\Lambda}$ as above, but *square*, $\mathrm{q} \times \mathrm{q}$

and then only first q columns of $\mathbf{U}$ are taken: which means that $\mathbf{U}$ has orthonormal columns, $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}$, but is not orthogonal, as $\mathbf{U}\mathbf{U}^{\mathsf{T}}$ may differ from $\mathbf{I}$

- if $\mathrm{p} \leqslant \mathrm{q}$, then the other way round: $\boldsymbol{\Lambda}$ is $\mathrm{p} \times \mathrm{p}$, $\mathbf{U}$ is square $\mathrm{p} \times \mathrm{p}$ and thus orthogonal, and $\mathbf{V}$ has orthonormal columns, $\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{I}$

Let us test it

```
> sa$u %*% t(sa$u)
           [,1]         [,2]         [,3]
[1,] 0.0925926  0.25925926  0.12962963
[2,] 0.2592593  0.92592593 -0.03703704
[3,] 0.1296296 -0.03703704  0.98148148
> t(sa$u) %*% sa$u
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> sa$v %*% t(sa$v)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> t(sa$v) %*% sa$v
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Full ("business class") version in R

```
> sa=svd(A,nu=dim(A,1))
> sa
$d
[1] 7.6203733 0.9643188

$u
           [,1]        [,2]       [,3]
[1,] -0.2932528 -0.08121183 -0.9525793
[2,] -0.7017514 -0.65838502  0.2721655
[3,] -0.6492670  0.74828724  0.1360828

$v
           [,1]       [,2]
[1,] -0.4782649  0.8782156
[2,] -0.8782156 -0.4782649
```

Testing...

```
> t(sa$u) %*% sa$u
              [,1]          [,2]          [,3]
[1,]  1.000000e-00  0.000000e+00 -9.714451e-17
[2,]  0.000000e+00  1.000000e+00 -1.387779e-16
[3,] -9.714451e-17 -1.387779e-16  1.000000e+00
> t(sa$v) %*% sa$v
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> sa$v %*% t(sa$v)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> sa$u %*% diag(sa$d) %*% t(sa$v)
Error in sa$u %*% diag(sa$d) : non-conformable arguments
> sa$u %*% rbind(diag(sa$d),c(0,0)) %*% t(sa$v)
     [,1] [,2]
[1,]    1    2
[2,]    2    5
[3,]    3    4
```

Mathematical Leftovers

Convexity

Probability Tidbits

Transformation of a density

Suppose that $\mathbf{X}$ is a p-dimensional random vector with density $g(\mathbf{x})$, and let $\mathbf{Y} = T(\mathbf{X})$, where T is a mapping from $\mathbb{R}^p$ to $\mathbb{R}^p$ possessing an inverse T^{-1}. If $\mathbf{X}$ has a probability density (with respect to the Lebesgue measure on $\mathbb{R}^p$) $h(\mathbf{x})$, then Y has a density

$$h(T^{-1}(\mathbf{x}))|\det(J_{T^{-1}}(\mathbf{x}))| = \frac{h(T^{-1}(\mathbf{x}))}{|\det(J_T(\mathbf{x}))|}$$

where J_T denotes the Jacobi matrix consisting of partial derivatives $\dfrac{\partial T_i(\mathbf{x})}{\partial x_j}$

If $T(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, then $J_T = \mathbf{A}$

Statistical Notions Perhaps Not That Known

Invariance and equivariance

Question: do functions used in statistics behave well when their arguments, say, change units? General problem: how these functions change under various transformations of their arguments? Sometimes they are

invariant: do not change under transformation

equivariant: do change under transformation, but appropriately

(We prefer to work with these notions intuitively rather than in a stricxt mathematical formalism)

Examples

Mean or median are equivariant under shifts: if we add b to all their arguments, the mean or median also increase by b

Mean or median are also equivariant under the affine transformation: if we change each x to $ax + b$, they change in the same way

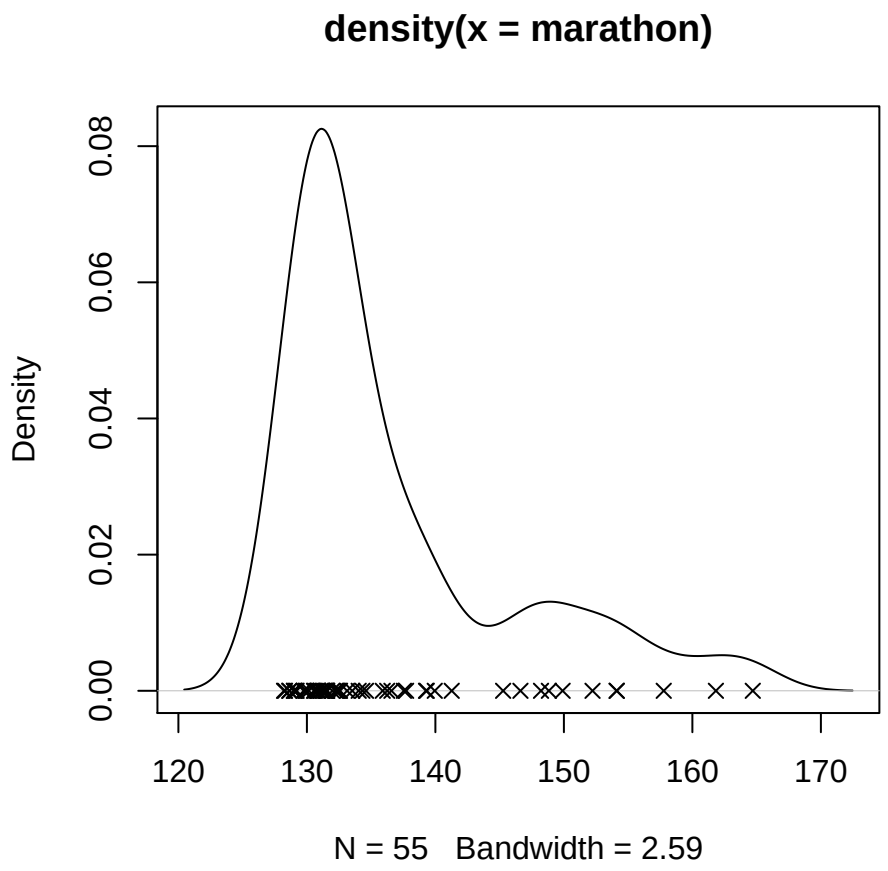
The standard deviation is equivariant under the scale change - if we multiply all arguments by $a > 0$, then standar deviation also gets multiplied by a - but it is invariant under shifts: if we add b to all arguments, standard deviation remains unchanged

The issue with mathematical formalization is that more sophisticated behavior is possible: for instance, the variance is still invariant under shifts, but it equivariance behavior under scale change is different: if each argument gets multiplied by $a > 0$, the variance gets multiplied by a^2

Nonparametric Univariate Statistics Recalled: Kernel Density Estimation

The probability density can be estimated?

```
> attach(Trackmen)
> plot(density(marathon))
> points(marathon,rep(0,length(marathon)),pch=4)
```

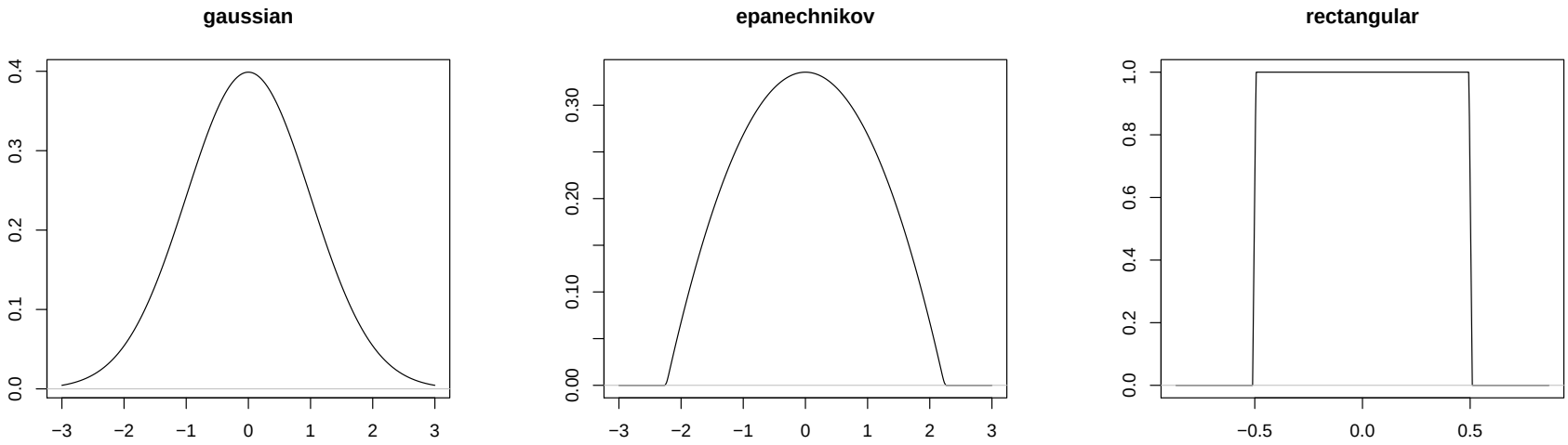


Kernel density estimator

$$\widehat{f}(x) = \frac{1}{nb} \sum_i K\left(\frac{x_i - x}{b}\right)$$

kernel: $\int K(u)\,du = 1$ and also $K(u) \geqslant 0$

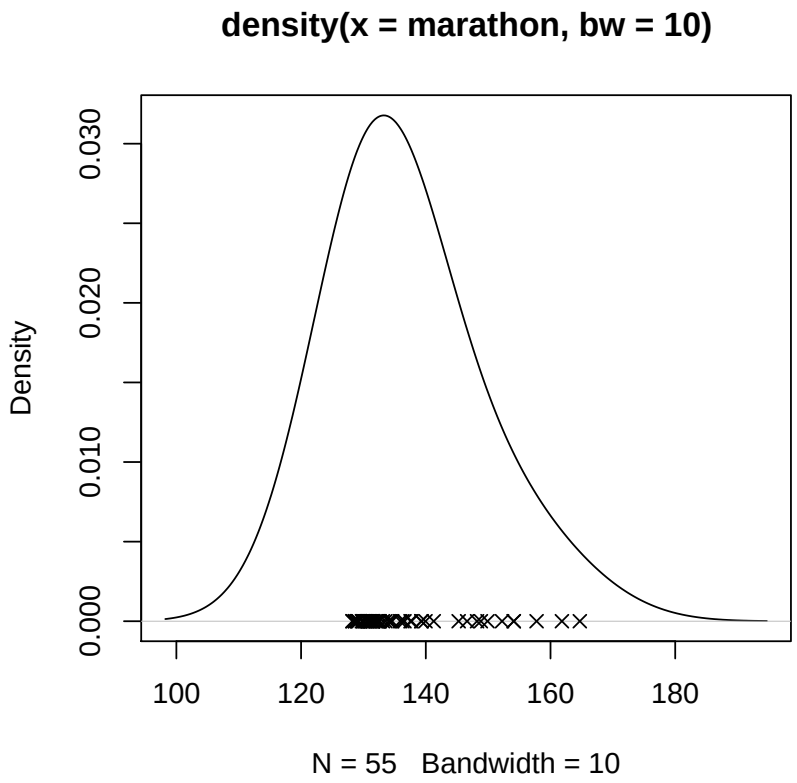
Examples: Gaussian (standard normal density), Epanechnikov, Rectangular (Parzen), and others

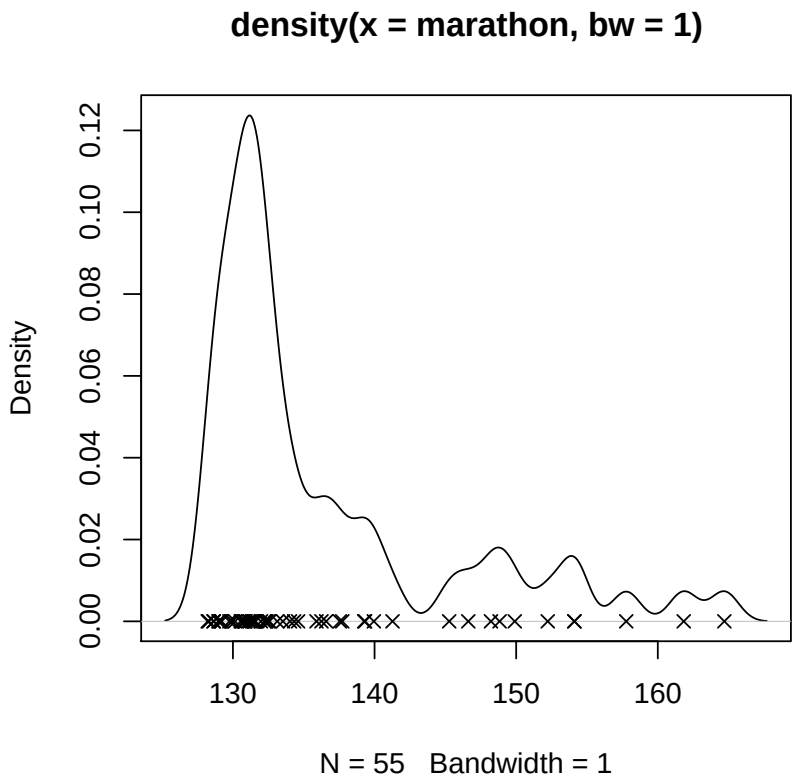


What does rectangular kernel mean? For $b = 1$, $\frac{1}{n}\sum_i K(x_i - x)$ is the relative proportion number of points falling into $[x - 1/2, x + 1/2]$; for general b, we obtain the relative proportion of points falling into $[x - b/2, x + b/2]$, divided by the length b of the interval.

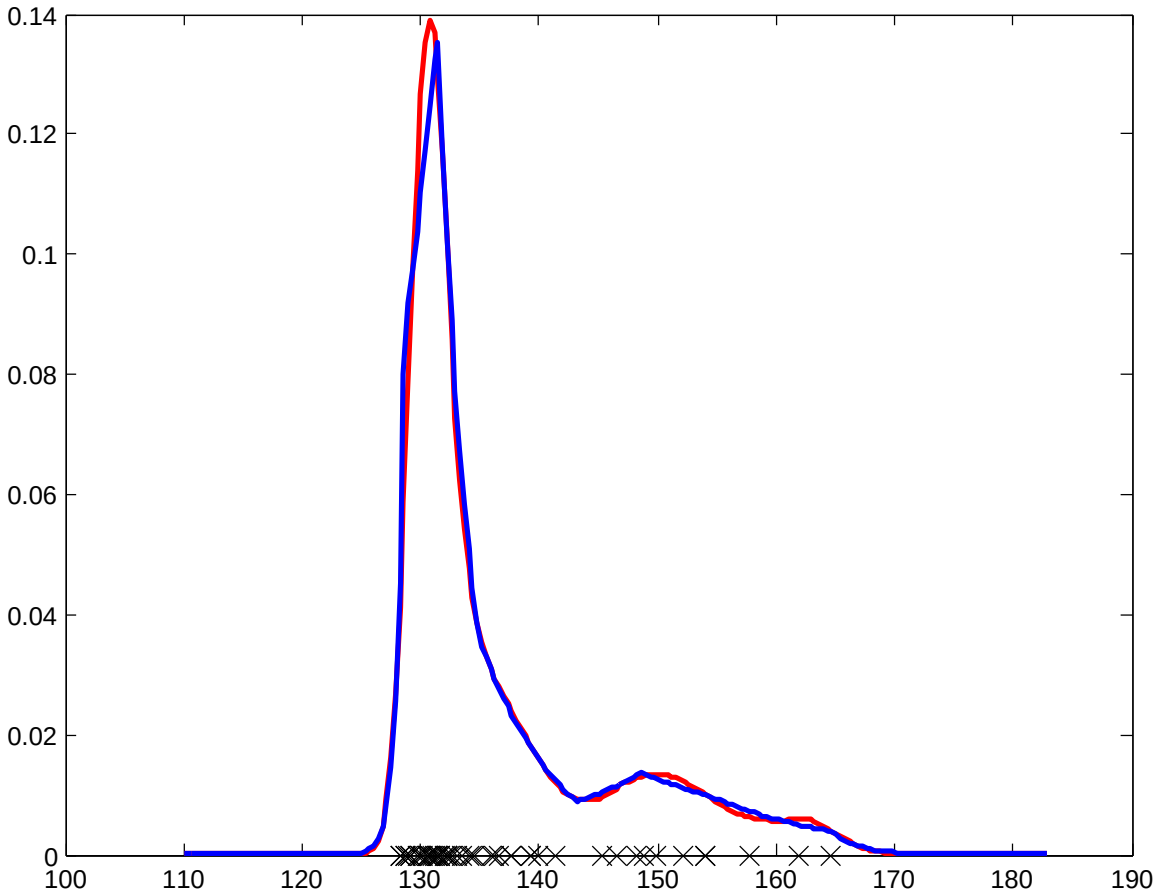
Different bandwidth

The same *bandwidth* $\mathfrak{b}$ may not equally adapt to all parts of the data





Note: there may be better estimators...



Nonparametric Bivariate Statistics Missed: Smoothing Splines And Some New Statistics Too

History

Whittaker (1923), "graduation" of actuarial mortality table

Given $y_1, y_2, \ldots, y_n$, find $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ such that

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum (\Delta^2 \hat{y}_i)^2 \curvearrowright \min_{\hat{y}} !$$

Here $\lambda \geqslant 0$. Objective: to rid the original data of fluctuations

General functional fitting

We now formulate the initial problem in a *functional*, that is, *infinite-dimensional* space

Given $y_1, y_2, \ldots, y_n$, and $x_1 < x_2 < \cdots < x_n$, find f such that

- $f(x_1), \ldots, f(x_n)$ fit well $y_1, y_2, \ldots, y_n$

- but at the same time, f is not too "wiggly", not too "rough"

(Some simplification here. We assume that no two x_i's coincide, which in practice may happen: we can have several observations with the same x_i. In practice this makes no problem, but the exposition is simpler without that)

Note first that least-squares or any other measure of lack-of-fit does not yield an unambiguous solution:

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 = 0 \qquad \text{when } y_i = f(x_i)$$

- and there are many such functions

Roughness penalty

The way out is to propose some measure of "wiggliness":

- take some derivative f the fitted function: f', or f'', or f'''

- then take its absolute value or square (magnitudel; sign is irrelevant)

- and finally make it global: integrate over x

For instance, $J(f) = \int (f''(x))^2 dx$; or $J(f) = \int |f''(x)| dx$

Such $J(f)$ will be referred to as (roughness) *penalty*.

To gain some partial insight about such a penalty, it may be instructive to investigate which f satisfy $J(f) = 0$; for both examples of J given above, we obtain that f is linear, $f(x) = \alpha + \beta x$.

And now, how to employ this penalty:

Penalized fitting

We can seek a fit with guaranteed wiggliness

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 \looparrowright \min_f ! \qquad J(f) \leqslant \Lambda$$

which via Lagrange multiplier theory (but for inequalities, not equalities!) is equivalent to

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda J(f) \looparrowright \min_f !$$

in the following sense: for any *tuning constant* Λ there is another tuning constant $\lambda > 0$, with unambiguous (but typically not explicit) relationship to Λ

Schoenberg (1964): smoothing splines

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int (f''(x))^2 dx \looparrowright \min_f !$$

There are mathematical details here, which we omit. However the question is: why splines?

Because the solutions are splines

The solution of the smoothing spline problem is a *natural* cubic spline, with knots at x_i (and only x_i)

Also, "natural": it just says that outside of knots it continues linearly. The first two derivatives (for a cubic spline) are to be matched: that is, at the extremal knots the first derivative, and also the second one, which is zero (second derivative of a linear function)

Note: once $f(x_i)$ given, the solution is found by minimizing $J(f)$

That gives the linearity of f outside the knots; inside of the knots, some further mathematics (it may be simply integration by parts) shows that...

... the solution to the smoothing spline problem, exists within the class of natural cubic splines, with knots at x_i (and only at x_i)

Finitary perspective

The original problem acted in the general functional spaces; now, however, we are in the finite dimensional space: all natural splines with given knots (finite number of knots, right?) can be written as linear combination of some (finite) basis functions

$$f(x) = \sum_j b_j g_j(x)$$

With some skill, we rewrite everything as a finite-dimensional problem in b_j - and in fact a quadratic one, as

- we are doing least-squares fitting) problem

- and the penalty has some square in it too, so it can be written as a quadratic form in b_j

And finally it is easy

So the original $$\sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int_c^d (f''(x))^2 dx \to \min_f !$$

becomes $$(y - Lb)^T(y - Lb) + \lambda b^T G b \to \min_b !$$

where L is a linear operator (=matrix) yielding the functional values at the x_i's in terms of the b_j's, and G defines a quadratic form related to the penalty

(the solution in fact solves the system $L^T L + \lambda G b = L^T y$)

Remarks

The selection of the basis does not play a role, as long as the bases are equivalent (they generate the same linear spaces, any function that is a linear combination in one base, is a linear combination in another one)

Thus, we have something more general than just bases here...

The aforementioned technical issue: if $\mathbf{x}_i$ have duplicate values among them, we should take some care; there is no problem in the first, lack-of-fit part of the objective function, but the second, penalty part, should involve only "cleaned" $\mathbf{x}_i$, with duplicates removed.

Regularization

One can use splines in regression in several ways:

- for instance, take the class of splines with several fixed knots and estimate their coefficients by simple least-squares fitting - which makes no problems if the number of fitted parameters is less than n (this is referred to as *regression splines*)

- or take many knots - possibly every x_i is a knot - then stipulate that the fit is a spline (that is, specify a basis and continue in finitary manner); we have now too many parameters, but we can still use (Tikhonov) *regularization* via a penalty (note that it is a different one):

$$\sum_{i=1}^{n} (y_i - \sum_j \beta_j g_j(x_i))^2 + \lambda \sum_j \beta_j^2 \looparrowright \min_{\beta} !$$

(this is referred to as *penalized splines*)

- or we can do it as above: formulate it in the infinite-dimensional space of funtions and only then take the finitary interpretation via so-called *representer theorem smoothing splines*)

Regularization: not only splines

In classical least squares regression estimation we seek $\boldsymbol{\beta}$ minimizing least squares lack-of-fit criterion $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathsf{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$

The necessary condition to obtain a solution is that $\mathbf{X}$ has full rank, or, equivalently $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ is invertible. What to do if this is violated, either approximately or exactly: for instance, when the number of predictors exceeds the number of datapoints: $\mathfrak{p} > \mathfrak{n}$?

We can replace the simple minimization of sum of squares by a penalized one: seek $\boldsymbol{\beta}$ minimizing

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathsf{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^{\mathsf{T}}\boldsymbol{\beta}$$

This is referred to as *ridge regression* (originally proposed for the problems when the matrix $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ has problems with invertibility), and is a special case of (Tikhonov) regularization

The penalty being the square of the ℓ^2 norm makes it quite amenable to solving via linear equations: note that the matrix $(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I})$ is invertible for any $\lambda > 0$, yielding the solution

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} = \mathbf{X}^{\mathsf{T}}\mathbf{y}$$

Atomic pursuit (LASSO)

So, regularization prescriptions construct fits by trading off between (a general) lack-of-fit criterion and (also general) penalty. The extent of this trade-off is controled by *smoothing*, *regularization parameter* λ.

An instance of this generality is an alternative of the ridge regression called *atomic pursuit*, more widely known as LASSO: instead of the ℓ^2 one uses the ℓ^1 penalty; if $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)^\top$, the minimized function is

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \sum_j |\beta_j|$$

Why such an alternative? While for $\lambda > 0$ it still handles situations when $\mathbf{X}$ is not of full rank (for instance when $p > n$), the absolute value in the penalty causes the resulting vector $\boldsymbol{\beta}$ of estimates to be *sparse* - to contain only few nonzero elements

This is unlike the ridge regression, which returns solutions that are rather nonzero; even for the regressors that do not have predictive value for the response, it tends to return estimates that are small in magnitude, but still not exactly zero

Another instance: tree-based methods

There is also an instance in tree-based methods (compare later): the function to be minimized is

$$\mathrm{R} + \alpha \text{ size}$$

where

- R is the lack-of-fit criterion here
- size is the complexity measure (penalty)
- and λ is called α instead

The catch: the tuning knob of regularization

Regularization prescriptions thus construct fits by trading off between (a general) lack-of-fit criterion and (also general) penalty. The extent of this trade-off is controled by *smoothing*, *regularization parameter* λ.

It is a *tuning* parameter: if we go back to the original formulation of smoothing splines, we notice

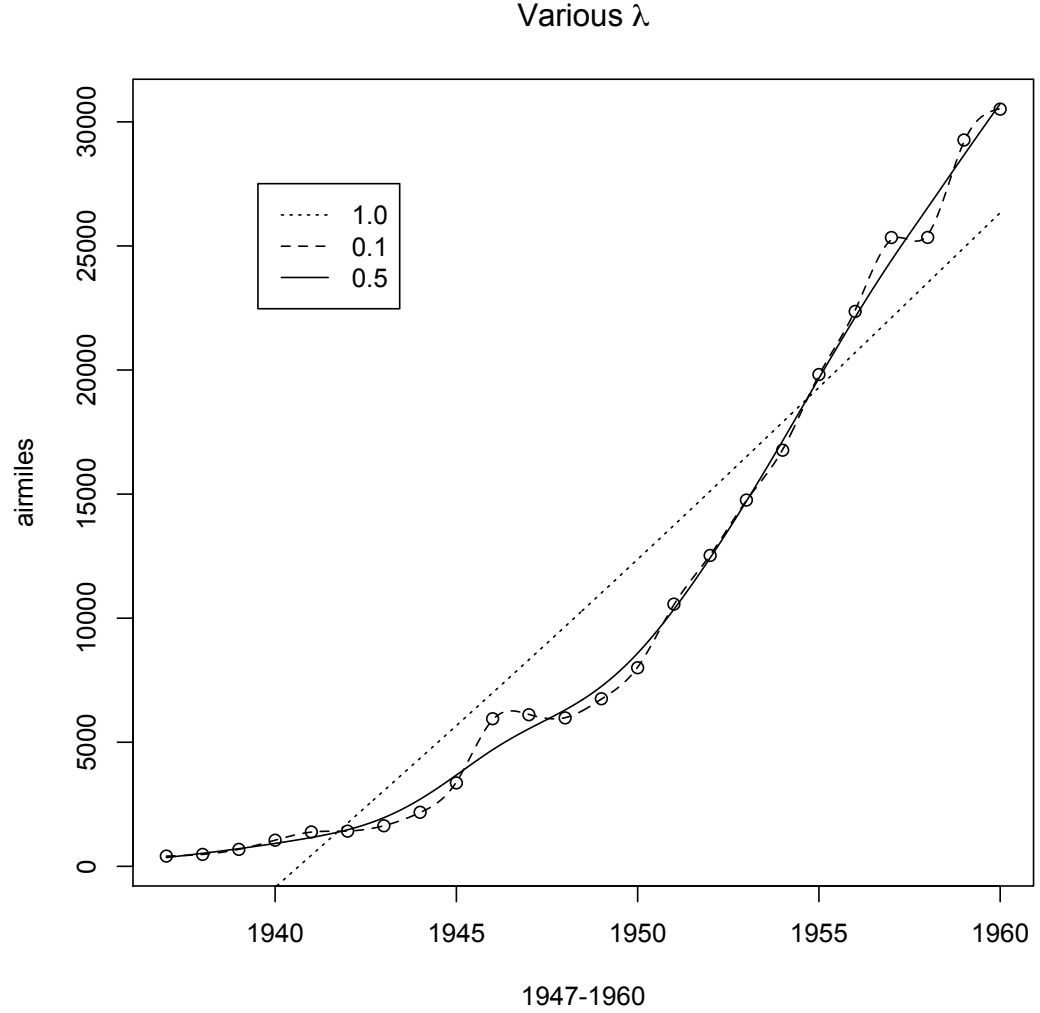
- for large λ the penalty prevails: the fit is linear

- for $\lambda \to 0$ ($\lambda = 0$ won't fly!) the lack-of-fit prevails: the fit, if there are no duplicates in the x_i's, is just the spline interpolation of the data

Of course, this is nice - but what is the *right* λ??

(Several ways to tackle this question...)

Revenue passenger airmiles flown by US airlines




```
> legend(locator(),lty=c(3,2,1),legend=c('1.0','0.1','0.5'))
```

The `legend` command does not show λ but `spar`. A closer look at `help(smooth.spline)` reveals that `spar` is a monotonous function of λ, normed so that `spar` lies between 0 and 1.

Finesses of the R implementation I

```
> plot(1937:1960,airmiles,xlab='1947-1960')
> title(expression(paste("Various ",lambda)))
> xx=seq(1937,1960,len=400)
> smsp=smooth.spline(1937:1960,airmiles,spar=1)
> lines(xx,predict(smsp,xx)$y,lty=3)
> smsp
Call:
smooth.spline(x = 1937:1960, y = airmiles, spar = 1)
Smoothing Parameter  spar= 1  lambda= 0.9681153
Equivalent Degrees of Freedom (Df): 2.063613
Penalized Criterion: 197298405
GCV: 9840216
```

Finesses of the R implementation II

```
> smsp=smooth.spline(1937:1960,airmiles,spar=.1)
> lines(xx,predict(smsp,xx)$y,lty=2)
> smsp
Call:
smooth.spline(x = 1937:1960, y = airmiles, spar = 0.1)
Smoothing Parameter  spar= 0.1  lambda= 3.045644e-07
Equivalent Degrees of Freedom (Df): 22.97617
Penalized Criterion: 34624.87
GCV: 792768.6

> smsp=smooth.spline(1937:1960,airmiles,spar=.5)
> lines(xx,predict(smsp,xx)$y)
> smsp
Call:
smooth.spline(x = 1937:1960, y = airmiles, spar = 0.5)
Smoothing Parameter  spar= 0.5  lambda= 0.0002363563
Equivalent Degrees of Freedom (Df): 7.460656
Penalized Criterion: 6128442
GCV: 537681.1
```