In the first part of the course on linear models, we’ve seen how to construct a linear model when the vector of covariates \boldsymbol{x} is given, so that \mathbb{E}(Y|\boldsymbol{X}=\boldsymbol{x}) is either simply \boldsymbol{x}^\top\boldsymbol{\beta} (for standard linear models) or a functional of \boldsymbol{x}^\top\boldsymbol{\beta} (in GLMs). But more generally, we can consider transformations of the covariates, so that a linear model can be used. In a very general setting, consider \sum\_{j=1}^m\beta\_j h\_j(\boldsymbol{x})with h\_j:\mathbb{R}^p\rightarrow\mathbb{R}. The standard linear model is obtained when m=p and h\_j(\boldsymbol{x})=x\_j , but of course, much more general models can be obtained, for instance with h\_k(\boldsymbol{x})=x\_j^2 orh\_k(\boldsymbol{x})=x\_{j}x\_{j'}, that could be used to achieve high-order Taylor expansions. In that case, we will obtain the polynomial regression, that we will discuss first. We might also think of piecewise constant functions, h\_k(\boldsymbol{x})=\boldsymbol{1}(x\_j\in [a,b]) , that could be related to regression trees (but that is not in the scope in the STT5100 course). And if we go on step futher, we might think of piecewise linear or piecewise polynomial function, possibly with additional continuity constraints, that will lead us to spline basis.

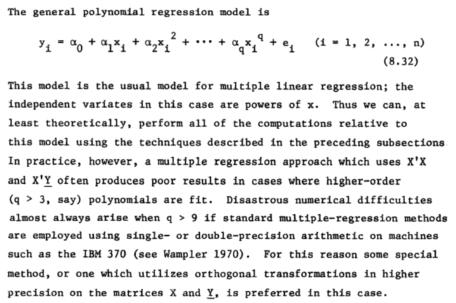
* **Polynomial regression**

For pedagogical purpose, when I talk about polynomial regression, I always have in mind (in the univariate case) y=\beta\_0+\beta\_1x+\beta\_2x^2+\cdots+\beta\_kx^k+\varepsilonbut if we use

|  |  |
| --- | --- |
| 1 | **lm**(y~**poly**(x,k)) |

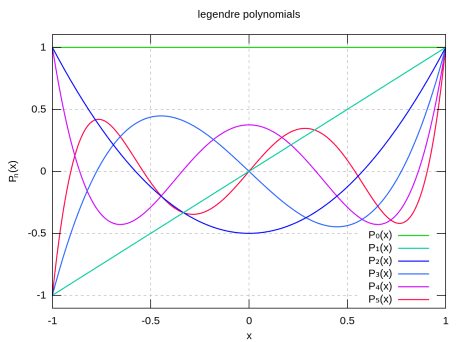
in R, the output is not the \beta\_j‘s.

As discussed in Kennedy & Gentle (1980) [*Statistical Computing*](https://www.worldcat.org/title/statistical-computing/oclc/6042923)*,*



Recall that orthogonal polynomials are defined with respect to the classical inner-product (on the finite interval (a,b)){\displaystyle \langle f,g\rangle =\int \_{a}^{b}f(x)g(x)~\mathrm {d} x} And a sequence of orthogonal polynomials is (P\_n) where P\_n is a polynomial of degree n, for all n, and such that P\_m\perpP\_n for all m\neq n. Note that those polyomials are orthogonal with respect to the inner product defined above, i.e. given some finite interval (a,b). But if (a,b) changes, the polynomials will be different.

A popular family of orthogonal polynomial, on finite interval (-1,+1) is the family of [Legendre polynomials,](https://en.wikipedia.org/wiki/Legendre_polynomials) satisfying{\displaystyle \int \_{-1}^{1}P\_{m}(x)P\_{n}(x)~\mathrm {d} x=0}as soon as m\neq n. Those polynomials satisfy Bonnet’s recursion formula{\displaystyle (n+1)P\_{n+1}(x)=(2n+1)xP\_{n}(x)-nP\_{n-1}(x)} or [Rodrigues’ formula](https://en.wikipedia.org/wiki/Rodrigues%27_formula) {\displaystyle P\_{n}(x)={\frac {1}{2^{n}n!}}{\frac {d^{n}}{dx^{n}}}(x^{2}-1)^{n}}The first values are here{\displaystyle P\_{0}(x)=1} {\displaystyle P\_{1}(x)=x}{\displaystyle P\_{2}(x)={\frac {3x^{2}-1}{2}}}{\displaystyle P\_{3}(x)={\frac {5x^{3}-3x}{2}}} {\displaystyle P\_{4}(x)={\frac {35x^{4}-30x^{2}+3}{8}}}

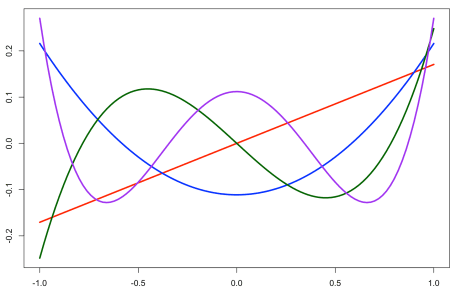


Interestingly, we can get those polynomial functions using

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **library**(orthopolynom)  (leg4coef = legendre.polynomials(n=4))  [[1]]  1    [[2]]  x    [[3]]  -0.5 + 1.5\*x^2    [[4]]  -1.5\*x + 2.5\*x^3    [[5]]  0.375 - 3.75\*x^2 + 4.375\*x^4 |

Of course, there are many families of orthogonal polynomials ([Jacobi polynomials](https://en.wikipedia.org/wiki/Jacobi_polynomials), [Laguerre polynomials](https://en.wikipedia.org/wiki/Laguerre_polynomials), [Hermite polynomials](https://en.wikipedia.org/wiki/Hermite_polynomials), etc). Now, in R, there is the standard poly function, that we use in polynomial regression.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | x = **seq**(-1,1,**length**=101)  y = **poly**(x,4)  y  1 2 3 4  [1,] -1.706475e-01 0.215984813 -2.480753e-01 0.270362873  [2,] -1.672345e-01 0.203025724 -2.183063e-01 0.216290298  ...  [100,] 1.672345e-01 0.203025724 2.183063e-01 0.216290298  [101,] 1.706475e-01 0.215984813 2.480753e-01 0.270362873  **attr**(,"coefs")  **attr**(,"coefs")$alpha  [1] 3.157229e-17 2.655145e-16 9.799244e-17 5.368224e-16    **attr**(,"coefs")$norm2  [1] 1.0000000 101.0000000 34.3400000 9.3377328 2.4472330 0.6330176    **attr**(,"degree")  [1] 1 2 3 4  **attr**(,"class")  [1] "poly" "matrix" |



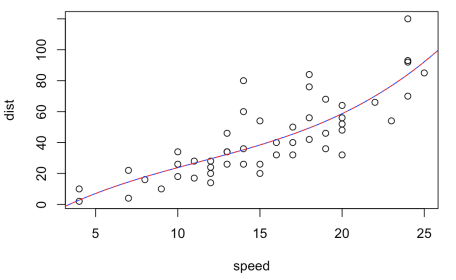
But these are not Legendre polynomials… As explained, the idea is to start with P\_{-1}(x)=0, P\_{0}(x)=1 and P\_{1}(x)=x, and then define \ell\_n=\langle P\_n,P\_n\rangle  as well as \alpha\_n=\langle P\_nP\_1,P\_1\rangle/\ell\_n=\langle P\_n^2,P\_1\rangle/\ell\_i= and \beta\_n=\ell\_n/\ell\_{n-1}. Finally, define recursively{\displaystyle P\_{n}(x)=(x-\alpha\_{n-1})P\_{n-1}(x)-\beta\_{i-1}P\_{i-2}(x)}and its normalized version, \tilde{P}\_{n}=P\_n/\sqrt{\ell\_n}. That is what poly computes.

So, for pedagogical purpose, I said that I like to use y=\boldsymbol{x}^\top\boldsymbol{\beta}+\varepsilon where\boldsymbol{x}=(1,x,x^2,\cdots,xˆ{k-1},x^k)And actually, when using poly, we use the [QR decomposition](https://en.wikipedia.org/wiki/QR_decomposition) of that matrix. As discussed in in [李哲源](https://stackoverflow.com/users/4891738/%E6%9D%8E%E5%93%B2%E6%BA%90)‘s post, we can almost reproduce the poly function using

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | my\_poly - **function** (x, degree = 1) {  xbar = **mean**(x)  x = x - xbar  QR = **qr**(**outer**(x, 0:degree, "^"))  X = **qr.qy**(QR, **diag**(**diag**(QR$qr), **length**(x), degree + 1))[, -1, **drop** = FALSE]  X2 = X \* X  norm2 = **colSums**(X \* X)  alpha = **drop**(**crossprod**(X2, x)) / norm2  **beta** = norm2 / (**c**(**length**(x), norm2[-degree]))  **colnames**(X) = 1:degree  **scale** = **sqrt**(norm2)  X = X \* **rep**(1 / **scale**, each = **length**(x))  X} |

Nevertheless, the two models are equivalent. More precisely,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **plot**(**cars**)  reg1 = **lm**(**dist**~speed+**I**(speed^2)+**I**(speed^3),**data**=**cars**)  reg2 = **lm**(**dist**~**poly**(speed,3),**data**=**cars**)  u = **seq**(3,26,**by**=.1)  v1 = **predict**(reg1,newdata=**data.frame**(speed=u))  v2 = **predict**(reg2,newdata=**data.frame**(speed=u))  **lines**(u,v1,**col**="blue")  **lines**(u,v2,**col**="red",lty=2) |



We have exactly the same prediction here

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | v1[u==15]  121  38.43919  v2[u==15]  121  38.43919 |

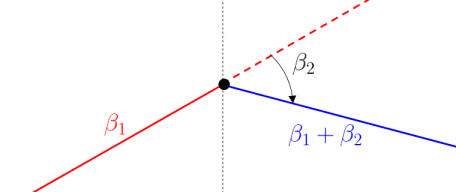
And probably also quite interesting : the coefficients do not have the same interpretation (since we do not have the same basis), but the p-value for the highest degree is exactly the same here ! Here the two models reject, with the same confidence, the polynomial of degree three,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **summary**(reg1)    Coefficients:  Estimate Std. Error **t** value Pr(&gt;|t|)  (Intercept) -19.50505 28.40530 -0.687 0.496  speed 6.80111 6.80113 1.000 0.323  **I**(speed^2) -0.34966 0.49988 -0.699 0.488  **I**(speed^3) 0.01025 0.01130 0.907 0.369    Residual standard error: 15.2 on 46 degrees of freedom  Multiple R-squared: 0.6732, Adjusted R-squared: 0.6519  F-statistic: 31.58 on 3 and 46 DF, p-value: 3.074e-11    **summary**(reg2)    Coefficients:  Estimate Std. Error **t** value Pr(&gt;|t|)  (Intercept) 42.98 2.15 19.988 &lt; 2e-16 \*\*\*  **poly**(speed, 3)1 145.55 15.21 9.573 1.6e-12 \*\*\*  **poly**(speed, 3)2 23.00 15.21 1.512 0.137  **poly**(speed, 3)3 13.80 15.21 0.907 0.369  ---  Signif. **codes**: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1    Residual standard error: 15.2 on 46 degrees of freedom  Multiple R-squared: 0.6732, Adjusted R-squared: 0.6519  F-statistic: 31.58 on 3 and 46 DF, p-value: 3.074e-11 |

* **B-splines regression (and GAMs)**

Splines are also important in regression models, especially when we start talking about [Generalized Additive Models](https://en.wikipedia.org/wiki/Generalized_additive_model). for a review. In the univariate case, I introduce (linear) splines through positive parts, in the sense thaty=\beta\_0+\beta\_1x+\beta\_2(x-s\_1)\_++\cdots+\beta\_k(x-s\_{k-1})\_++\varepsilonwhere (x-s)\_+ equals 0 if x<s[/latex] and [latex]x-s[/latex] if [latex]x>s. Those functions are nice since they are continuous, so the model is continuous (the weighted sum of continuous functions is continuous). And we can go one step further, with y=\beta\_0+\beta\_1x+\beta\_2x^2+\beta\_3(x-s\_1)^2\_++\cdots+\beta\_k(x-s\_{k-2})^2\_++\varepsilonwith quadratic splines, or y=\beta\_0+\beta\_1x+\beta\_2x^2+\beta\_3x^3+\beta\_4(x-s\_1)^3\_++\cdots+\beta\_k(x-s\_{k-3})^3\_++\varepsilonfor cubic splines. Interestingly, quadratic splines are not only continuous, but their first derivative is also continuous (and the second one for cubic splines). So the knot discontinuity is s\_1,s\_2,\cdots is now invisible…

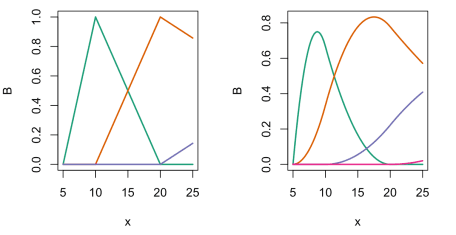
I like those models since they are easy to interprete. For example, the simple model \beta\_1 x+\beta\_2(x-s)\_+ is the following piecewise linear function, continuous, with a “rupture” at knot s.



Observe also the following interpretation: for small values of x, there is a linear increase, with slope \beta\_1, and for lager values of x, there is a linear decrease, with slope \beta\_1+\beta\_2. Hence, \beta\_2 is interpreted as a change of the slope.

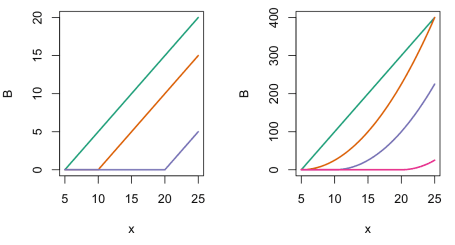
Unfortunately, it is now what R is using when using the bs function in R, which are the standard [B-splines](https://en.wikipedia.org/wiki/B-spline). Just to visualize (I will skip the maths here), with R, we have

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **library**(splines)  clr6 = **c**("#1b9e77","#d95f02","#7570b3","#e7298a","#66a61e","#e6ab02")  x = **seq**(5,25,**by**=.25)  B = bs(x,**knots**=**c**(10,20),Boundary.knots=**c**(5,55),degre=1)  **matplot**(x,B,type="l",lty=1,lwd=2,**col**=clr6)  B=bs(x,**knots**=**c**(10,20),Boundary.knots=**c**(5,55),degre=2)  **matplot**(x,B,type="l",**col**=clr6,lty=1,lwd=2) |



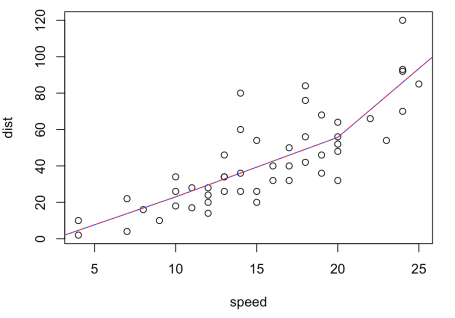
while the functions I mentioned were (more or less) the following

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | pos = **function**(x,s) (x-s)\*(x&gt;s)  **par**(mfrow=**c**(1,2))  clr6 = **c**("#1b9e77","#d95f02","#7570b3","#e7298a","#66a61e","#e6ab02")  x = **seq**(5,25,**by**=.25)  B = **cbind**(pos(x,5),pos(x,10),pos(x,20))  **matplot**(x,B,type="l",lty=1,lwd=2,**col**=clr6)  pos2 = **function**(x,s) (x-s)^2\*(x&gt;s)  B = **cbind**(pos(x,5)\*20,pos2(x,5),pos2(x,10),pos2(x,20))  **matplot**(x,B,type="l",**col**=clr6,lty=1,lwd=2) |



And as for the polynomial regression, the two models are equivalent. For example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **plot**(**cars**)  reg1 = **lm**(**dist**~speed+pos(speed,10)+pos(speed,20),**data**=**cars**)  reg2 = **lm**(**dist**~bs(speed,degree=1,**knots**=**c**(10,20)),**data**=**cars**)  v1 = **predict**(reg1,newdata=**data.frame**(speed=u))  v2 = **predict**(reg2,newdata=**data.frame**(speed=u))  **lines**(u,v1,**col**="blue")  **lines**(u,v2,**col**="red",lty=2) |



or more specifically

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | v1[u==15]  121  39.35747  v2[u==15]  121  39.35747 |

So one more time, the two models are equivalent, but I still find the approach with the positive part more intuitive, and easy to understand. As well as the interpretation of coefficients,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | **summary**(reg1)    Coefficients:  Estimate Std. Error **t** value Pr(&gt;|t|)  (Intercept) -7.6305 16.2941 -0.468 0.6418  speed 3.0630 1.8238 1.679 0.0998 .  pos(speed, 10) 0.2087 2.2453 0.093 0.9263  pos(speed, 20) 4.2812 2.2843 1.874 0.0673 .  ---  Signif. **codes**: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1    Residual standard error: 15 on 46 degrees of freedom  Multiple R-squared: 0.6821, Adjusted R-squared: 0.6613  F-statistic: 32.89 on 3 and 46 DF, p-value: 1.643e-11    **summary**(reg2)    Coefficients:  Estimate Std. Error **t** value Pr(&gt;|t|)  (Intercept) 4.621 9.344 0.495 0.6233  bs(speed, degree = 1, **knots** = **c**(10, 20))1 18.378 10.943 1.679 0.0998 .  bs(speed, degree = 1, **knots** = **c**(10, 20))2 51.094 10.040 5.089 6.51e-06 \*\*\*  bs(speed, degree = 1, **knots** = **c**(10, 20))3 88.859 12.047 7.376 2.49e-09 \*\*\*  ---  Signif. **codes**: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1    Residual standard error: 15 on 46 degrees of freedom  Multiple R-squared: 0.6821, Adjusted R-squared: 0.6613  F-statistic: 32.89 on 3 and 46 DF, p-value: 1.643e-11 |

Here we can see directly that the first knot was not interesting (the slope did not change significantly) while the second one was…