Lecture 28: Bernoulli Polynomials

This lecture introduces Bernoulli polynomials, which will be used to complete the proof that Trapezoidal rule converges like $O(n^{-2})$ for non-periodic functions and $O(n^{\alpha})$ for periodic functions. We will do so by proving the *Euler–MacLaurin formula*:

$$\int_0^1 f(x)dx - h\left(\frac{f(x_0)}{2} + \sum_{k=1}^{n-1} f(x_k) + \frac{f(x_n)}{2}\right) = -\sum_{j=2}^{\alpha} (-)^j \frac{B_j}{j!n^j}(f^{(j-1)}(1) - f^{(j-1)}(0)) + O(n^{-\alpha-1})$$

where B_j are constants defined in terms of the Bernoulli polynomials.

The key property about this error expression is that if the derivatives cancel, i.e., if $f^{(j-1)}(1) = f^{(j-1)}(0)$ for $j = 2, \ldots, \alpha$, then we have an increased order of convergence

$$\int_0^1 f(x)dx - h\left(\frac{f(x_0)}{2} + \sum_{k=1}^{n-1} f(x_k) + \frac{f(x_n)}{2}\right) = O(n^{-\alpha})$$

This explains why the Trapezium rule converges like $O(n^{-\alpha})$ for all α when f is periodic.

Before we get to Bernoulli, polynomials, we have some more programming topics to discuss.

Map

Map applies a function to each element in an `AbstractArray`:

In [65]:

```
f=x->x^2

map(f,[1.,2.,3.])
```

Out[65]:

```
3-element Array{Float64,1}:
 1.0
 4.0
 9.0
```

When used for `Matrix`, it preserves the shape:

In [66]:

```
map(f,[1. 2.; 3. 4.])
```

Out[66]:

```
2x2 Array{Float64,2}:
 1.0   4.0
 9.0  16.0
```

One particular use is for functions which do not extend naturally to Vector inputs:

In [67]:

```
function fwithif(x::Number)
    if -1 ≤ x ≤ 1
        0
    else
        x
    end
end
```

Out[67]:

```
fwithif (generic function with 2 methods)
```

In [68]:

```
map(fwithif,[1.,2.,3])
```

Out[68]:

```
3-element Array{Real,1}:
 0
 2.0
 3.0
```

Splat

Splat takes a vector/tuple and "splats" it out to become arguments of a function:

In [69]:

```
f=(x,y,z,w,t)->x+y+z
```

Out[69]:

```
(anonymous function)
```

In [70]:

```
v=[1.,2.,3.,4.,5.]
f(v...)
```

Out[70]:

```
6.0
```

Similar notation can be used for having an arbitrary number of function arguments, where in the following `x` becomes a `Tuple`:

In [74]:

```
function varlengthf(x...)
    # x is now a tuple, with length depending on number of inputs

    if length(x)≤2
        x
    else
        x[2]
    end
end
```

Out[74]:

```
varlengthf (generic function with 1 method)
```

In [75]:

```
varlengthf(1,5,3)
```

Out[75]:

```
5
```

In [79]:

```
varlengthf(3,4.0)
```

Out[79]:

```
(3,4.0)
```

One can combine normal arguments with splat arguments:

In [80]:

```
function varlengthg(x,y,z...)
    if length(z) == 0
        (x,y)
    else
        z
    end
end
```

Out[80]:

```
varlengthg (generic function with 1 method)
```

In [82]:

```
varlengthg(1)  # need at least an x and a y
```

```
LoadError: MethodError: `varlengthg` has no method matching varle
ngthg(::Int64)
Closest candidates are:
  varlengthg(::Any, !Matched::Any, !Matched::Any...)
while loading In[82], in expression starting on line 1
```

In [84]:

```
varlengthg(1,4.0)
```

Out[84]:

```
(1,4.0)
```

In [85]:

```
varlengthg(1,4.0,6.0,7.0,8.0)
```

Out[85]:

```
(6.0,7.0,8.0)
```

ApproxFun (Experimental, Don't use!)

`ApproxFun` is a Julia package

https://github.com/ApproxFun/ApproxFun.jl (https://github.com/ApproxFun/ApproxFun.jl)

for working with functions *numerically*: it makes it easy to do algebra/calculus/solve differential equations in an automatic fashion. We will use it to construct and plot Bernoulli polynomials, which are defined using indefinite integration.

Warning This package is experimental, and so its highly recommended that you don't use it for this class, unless you know what you are doing. It also doesn't work on the lab machines because of the firewall.

In `ApproxFun`, a `Fun` takes in a `Function` and an interval to create a numerical representation of the function. We can, for example, plot the function:

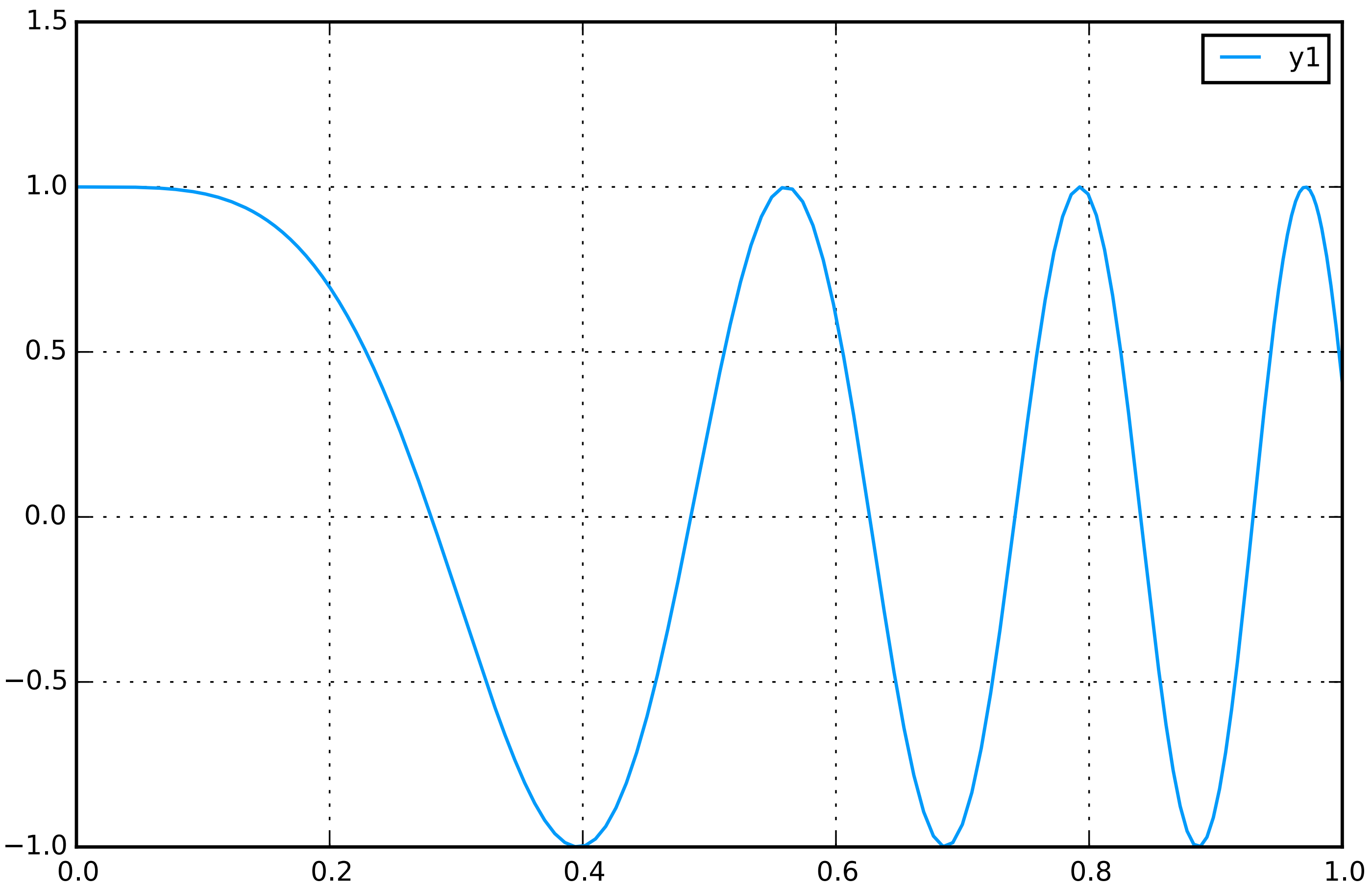
In [15]:

```
using ApproxFun

f=Fun(x->cos(20x^2),[0.,1.])

ApproxFun.plot(f)
```

Out[15]:



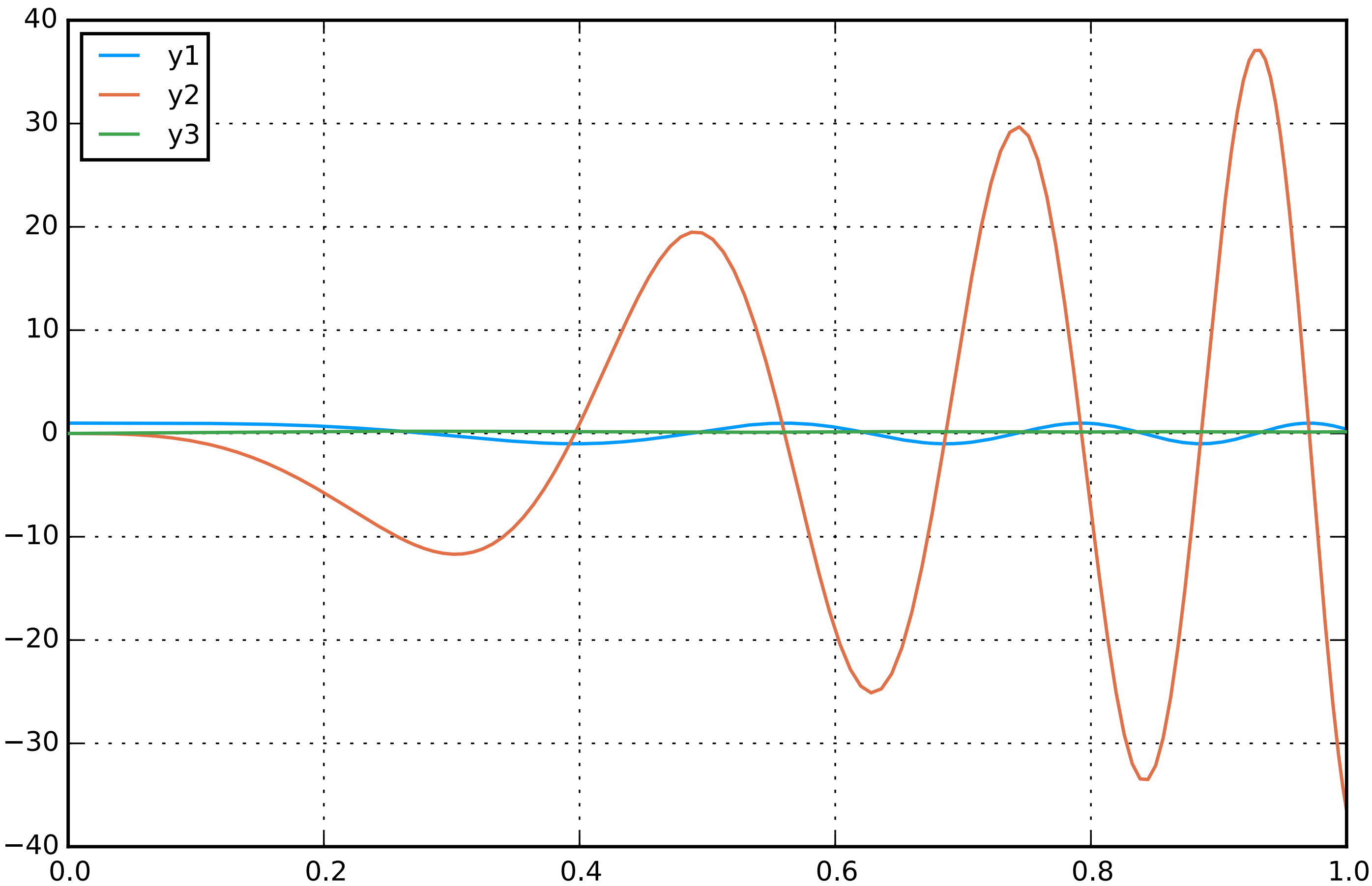
We can also integrate and differentiate:

In [2]:

```
sum(f)  # integral of f from [0,1.]

ApproxFun.plot([f,f',cumsum(f)])     # plot the function, its derivative and
 its indefinite integral
```

Out[2]:



Bernoulli Polynomials

Bernoulli polynomials are defined via the properties:

1. $B_0(x) = 1$
2. $B_k'(x) = kB_{k-1}(x)$ (Just like x^k)
3. $\int_0^1 B_k(x)dx = 0$ for $k = 1, 2, \ldots$

Note that

$$B_k(x) = B_k(0) + \int_0^x B_{k-1}(\chi)d\chi$$

A consequence of this fact and property 3 is that

$$B_k(1) = B_k(0) + \int_0^1 B_{k-1}(x)dx = B_k(0),$$

for $k = 2, 3, \ldots$. In other words, the Bernoulli polynomials are periodic!

We define the *Bernoulli numbers* by

$$B_k \triangleq B_k(1)$$

The first three can be easily worked out to be

$$B_0(x) = 1, B_1(x) = x - 1/2, B_2(x) = x^2 - x + 1/6.$$

Here's a plot:

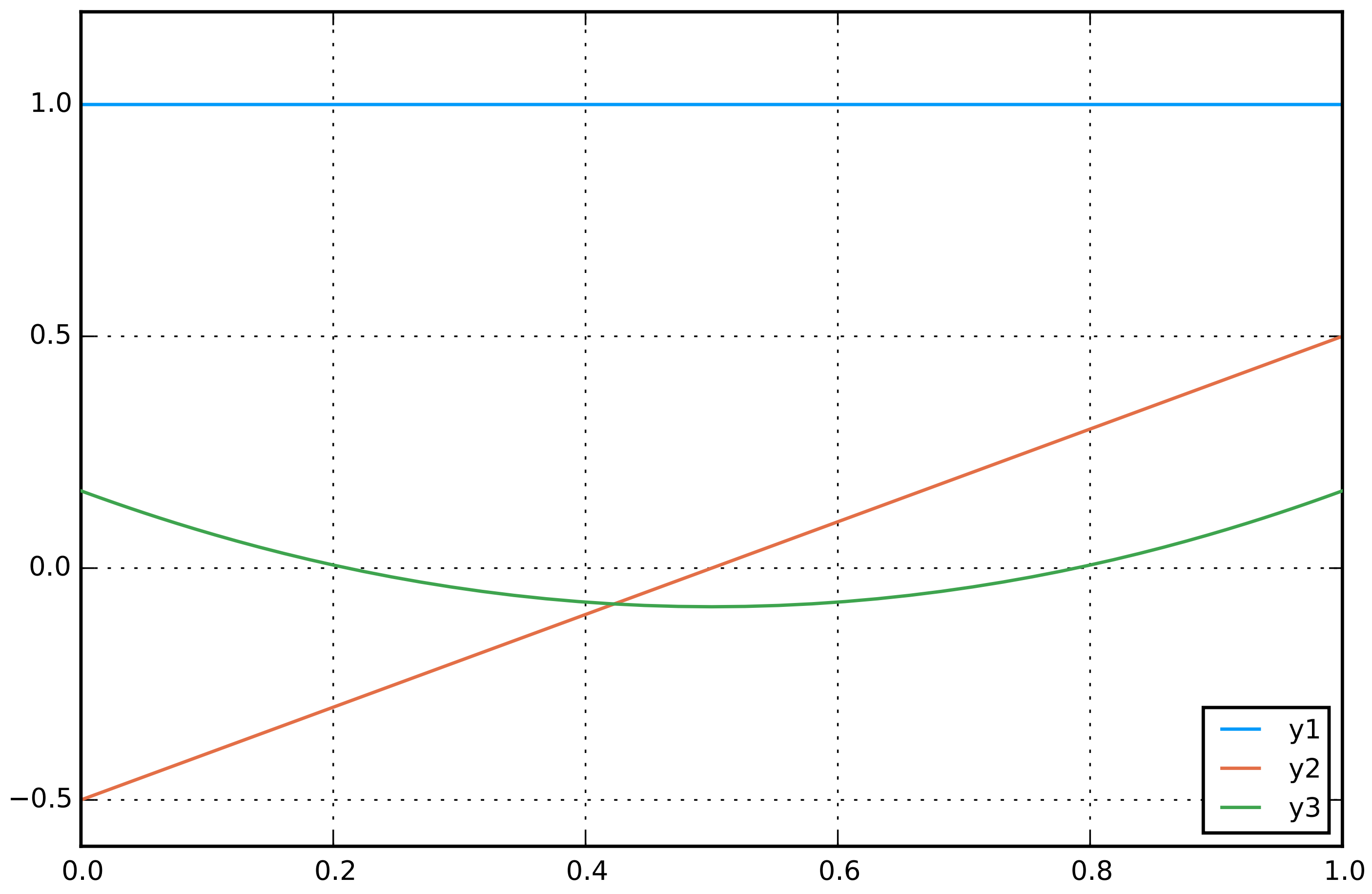
In [3]:

```
x=Fun([0.,1.])

B0=Fun(1,[0.,1.])
B1=Fun(x->x-1/2,[0.,1.])
B2=Fun(x->x^2-x+1/6,[0.,1.])


ApproxFun.plot([B0,B1,B2])
```

Out[3]:



We can check that B_1 and B_2 integrate to zero:

In [4]:

```
sum(B0),sum(B1),sum(B2)
```

Out[4]:

```
(1.0,0.0,-1.0408340855860843e-17)
```

And B_2 satisfies the periodicity property;

In [5]:

```
B2(1)-B2(0)
```

Out[5]:

```
0.0
```

We can now create B_3 by using indefinite integration. We want afunction that satisfies $B_3'(x) = 3B_2(x)$, so define

$$B_3(x) \triangleq 3 \int_0^x B_2(x) dx$$

This is done using `cumsum` in ApproxFun:

In [6]:

```
B3=3cumsum(B2)  # indefinite integral of B2


norm(3B2-B3')
```

Out[6]:

```
0.0
```

By chance, we also satisfy property 3: $\int_0^1 B_3(x)dx = 0$

In [7]:

```
sum(B3)  # integral is zero
```

Out[7]:

```
-9.107298248878237e-18
```

And indeed we have $B_3(1) = B_3(0) = B_3 = 0$:

In [8]:

```
B3(1),B3(0)
```

Out[8]:

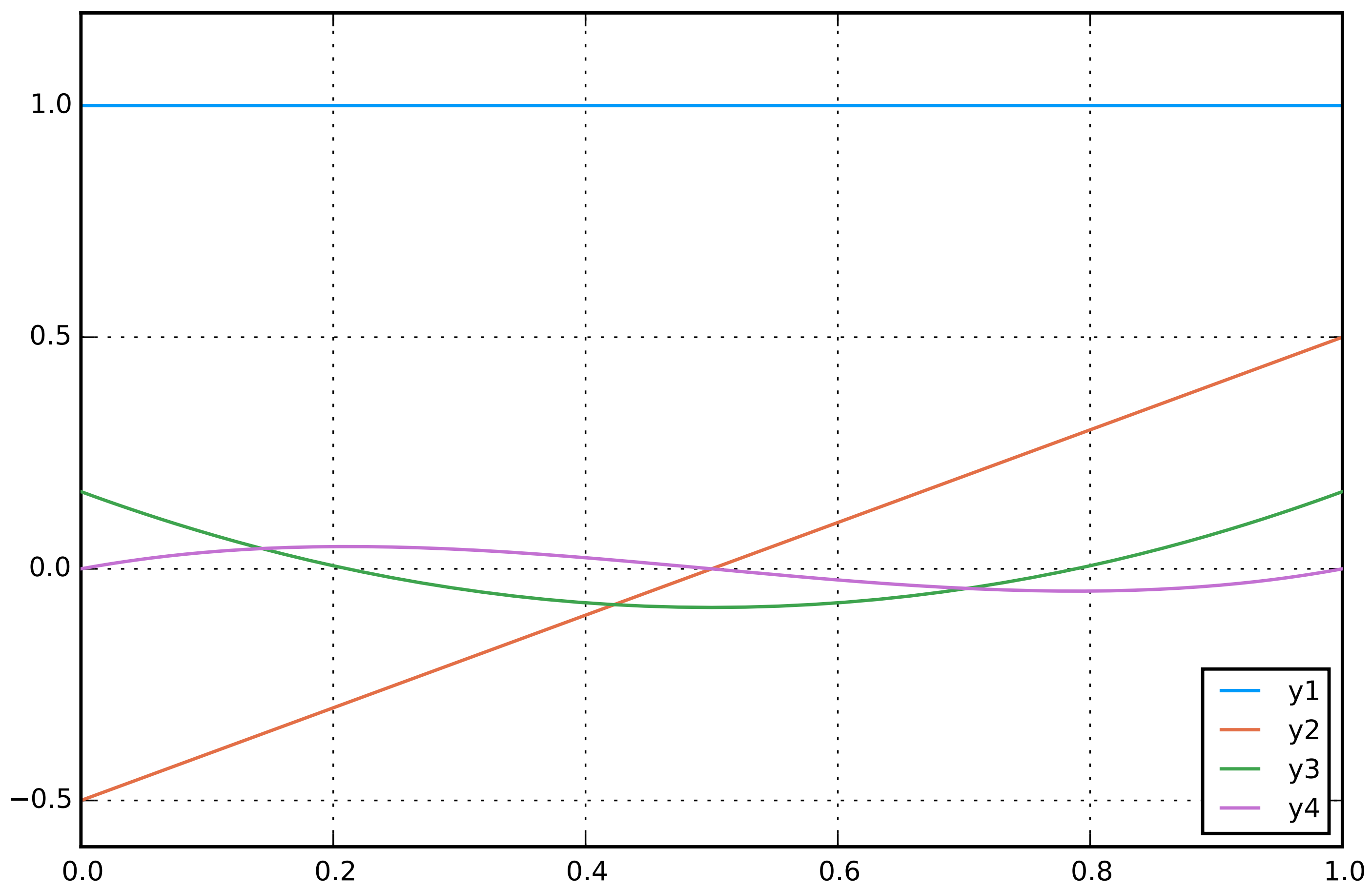
```
(-2.7755575615628914e-17,0.0)
```

Here's a plot of all our polynomials so far.

In [9]:

```
ApproxFun.plot([B0,B1,B2,B3])
```

Out[9]:



Let's try creating B_4 via indefinite integration, by defining

$$P_4(x) \triangleq 4 \int_0^x B_3(\chi) d\chi$$

In [10]:

```
B0=Fun(1,[0.,1.])
B1=Fun(x->x-1/2,[0.,1.])
B2=Fun(x->x^2-x+1/6,[0.,1.])

B3=3cumsum(B2)  # indefinite integral of B2

P4=4cumsum(B3)
norm(P4'-4B3)
```

Out[10]:

```
0.0
```

But now property 3 is not satisfied:

In [11]:

```
sum(P4)   # ≠ 0
```

Out[11]:

```
0.033333333333333326
```

We thus obtain B_4 by subtracting out the relevant constant:

$$B_4(x) \triangleq P_4(x) - \int_0^1 P_4(x)dx$$

In [12]:

```
B4=P4-sum(P4)
sum(B4)
```

Out[12]:

```
-8.131516293641283e-19
```

Indeed, we have $B_4(1) = B_4(0) = B_4(= -1/30)$:

In [13]:

```
B4(1),B4(0)
```

Out[13]:

```
(-0.03333333333333337,-0.033333333333333326)
```

We can continue the process, where the odd polynomials satisfy the periodicity for free:

In [14]:

```
B5=5cumsum(B4)

P6=6cumsum(B5)
B6=P6-sum(P6)

ApproxFun.plot([B0,B1,B2,B3,B4,B5,B6])
```

Out[14]: