# Computational Physics Homework 6

Cristopher Cerda Puga

# Richardson Extrapolation using the Euler-Maclaurin formula

It is of our interest to develop a method to extrapolate the value of the series:

$$\sum_{k=1} \frac{1}{k^s}$$

The partial sums of such series behave as:

$$\sum_{k=1}^{n} \frac{1}{k^s} \sim \zeta(s) - \frac{1}{(s-1)n^{s-1}} + \frac{1}{2n^s} - \sum_{i=1} \frac{B_{2i}}{(2i)!} \frac{(s+2i-2)!}{(s-1)! n^{s+2i-1}}$$

where the $B_{2i}$ correspond to the Bernoulli numbers. The strategy to solve this problem is to write a system of equations for multiple partial sums:

$$\begin{aligned}
A_N &= q_0 + \frac{q_1}{N^{s-1}} + \frac{q_1}{N^s} - \sum_{i=1} \frac{B_{2i}}{(2i)!} \frac{(s+2i-2)!}{(s-1)! N^{s+2i-1}} \\
A_{N+1} &= q_0 + \frac{q_1}{(N+1)^{s-1}} + \frac{q_1}{(N+1)^s} - \sum_{i=1} \frac{B_{2i}}{(2i)!} \frac{(s+2i-2)!}{(s-1)!(N+1)^{s+2i-1}} \\
&\dots \\
A_{N+p} &= q_0 + \frac{q_1}{(N+p)^{s-1}} + \frac{q_1}{(N+p)^s} - \sum_{i=1} \frac{B_{2i}}{(2i)!} \frac{(s+2i-2)!}{(s-1)!(N+p)^{s+2i-1}}
\end{aligned}$$

Or in matrix notation:

$$\begin{pmatrix} A_N \\ A_{N+1} \\ \dots \\ A_{N+p} \end{pmatrix} = \begin{pmatrix} 1 & 1/N^{s-1} & 1/N^s & \dots \\ 1 & 1/(N+1)^{s-1} & 1/(N+1)^s & \dots \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1/(N+p)^{s-1} & 1/(N+p)^s & \dots \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_p \end{pmatrix}$$

We will see in a moment how to get the dot columns in the matrix. For now, the strategy is as follows: we have a matrix equation

$$A = MQ$$

Where the only thing that is of our interest is the first entry of $Q$, which corresponds to the approximation of the series. Now, one way to get such vector is by calculating the inverse $M^{-1}$of $M$ and apply it to both sides of the matrix equation:

$$M^{-1}A = Q$$

However, it is difficult to find and explicit formula for the inverse, specially because since the fourth column the entries take a form related to the ith dependent terms of the Euler-Maclaurin formula. So, we must calculate the partial sums $A_i$, $i = N, \dots, N+p$, so increasing the number of $p$ implies increasing the size of the matrix $M$. We will do it numerically using `mpmath`, first up to $A_{N+3}$ and then up to $A_{N+4}$.

Now we can talk about the ith dependent terms of the matrix, for the $N+3$ case:

$$M = \begin{pmatrix} 1 & 1/N^{s-1} & 1/N^s & 1/N^{s+1} \\ 1 & 1/(N+1)^{s-1} & 1/(N+1)^s & 1/(N+1)^{s+1} \\ 1 & 1/(N+2)^{s-1} & 1/(N+2)^s & 1/(N+2)^{s+1} \\ 1 & 1/(N+3)^{s-1} & 1/(N+3)^s & 1/(N+3)^{s+1} \end{pmatrix}$$

Now we can start working on the problem

```
import mpmath as mp

mp.dps = 100

def partial(nmax, s):
    return mp.nsum(lambda n: mp.power(n, -s), [1, nmax])

A = mp.matrix([0,0,0,0])

k = 100 # here k corresponds to N in our previous analysis
s = 3/2 # the value of s

for i in range(len(A)):
    # add the values of the partial function to the matrix A
    nmax = i+k
    A[i] = partial(nmax, s)

# define the matrix M
M = mp.matrix([[1,1/k**(s-1),1/k**s,1/k**(s+1)],[1,1/(k+1)**(s-1),1/(k+1)**s,1/(k+1)↩
    **(s+1)],[1,1/(k+2)**(s-1),1/(k+2)**s,1/(k+2)**(s+1)],[1,1/(k+3)**(s-1),1/(k+3)**↩
    s,1/(k+3)**(s+1)]])

# calculate the inverse of M
Minv = mp.inverse(M)

# calculate Q matrix
Q = Minv*A
print(Q)
>>> [  2.61237534844416] -> The approximaton of the Riemann zeta function at s = 3/2
    [ -1.99999999491331]
    [ 0.499999550103022]
    [-0.124979896719848]
```

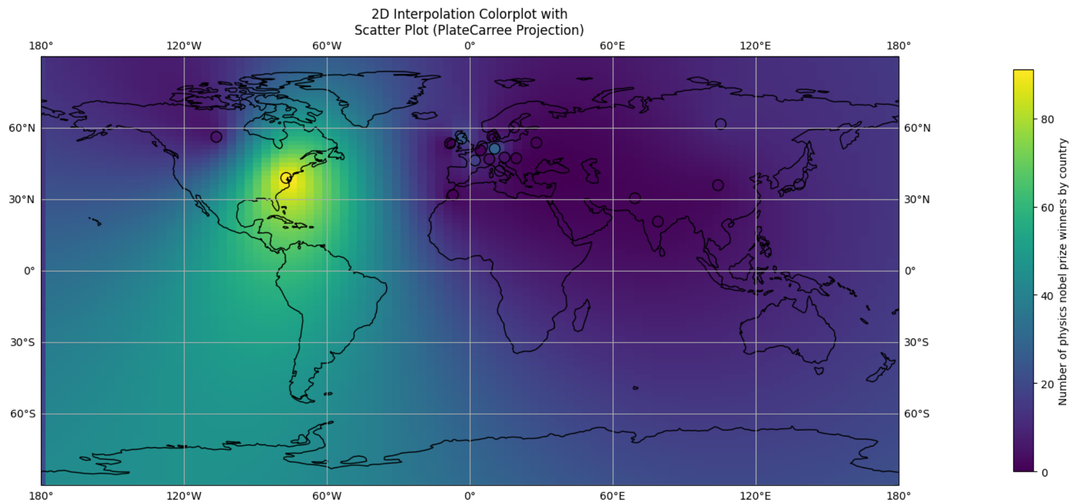
We can see that the approximation is pretty good. For a better approximation we can increase the size of our matrix and in consequence of the vector containing the partial sums, as well as select a good choose of k:

```
import mpmath as mp

mp.dps = 100
k = 50
s = 3/2

def partial(nmax, s):
    return mp.nsum(lambda n: mp.power(n, -s), [1, nmax])

B = mp.matrix([0,0,0,0,0])
for i in range(len(B)):
    nmax = i+k
    B[i] = partial(nmax, s)

T = mp.matrix([[1, 1/k**(s - 1), 1/k**(s), 1/k**(s + 1),1/k**(s + 3)],[1, 1/(k + 1)↩
    **(s - 1), 1/(k + 1)**(s), 1/(k + 1)**(s + 1), 1/(k + 1)**(s + 3)],[1, 1/(k + 2)↩
    **(s - 1), 1/(k + 2)**(s), 1/(k + 2)**(s + 1),1/(k + 2)**(s + 3)], [1, 1/(k + 3)↩
    **(s - 1), 1/(k + 3)**(s), 1/(k + 3)**(s + 1), 1/(k + 3)**(s + 3)],[1,1/(k + 4)↩
    **(s - 1), 1/(k + 4)**(s), 1/(k + 4)**(s + 1), 1/(k + 4)**(s + 3)]])
Tinv = mp.inverse(T)

Q2 = Tinv*B
print(Q2)
>>> [  2.61237534868775] -> The approximaton of the Riemann zeta function at s = 3/2
```

```
[   -2.0000000001121]
[ 0.500000005620954]
[-0.125000118835117]
[0.0182315027087133]
```

So we can see that for a very slow value of $p$, like 4 or 5, we get a pretty good approximation of the series. Increasing the size of the matrix (i.e. increasing $p$) gives us a better aproximation.

## Radial basis functions interpolation.




```python
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from scipy.interpolate import Rbf

lats = [38.895,51.165691,55.378051,46.227638,61.52401,36.204824,52.132633,56.130366,
        41.87194,60.128161,47.516231,46.818188,56.26392,35.86166,31.791702,53.41291,
        20.593684,30.375321,50.503887,53.709807,47.162494]
lons = [-77.0366,10.451526,-3.435973,2.213749,105.318756,138.252924,5.291266,
        -106.346771,12.56738,18.643501,14.550072,8.227512,9.501785,104.195397,
        -7.09262,-8.24389,78.96288,69.345116,4.469936,27.953389,19.503304]
data = [96,28,25,16,11,10,9,5,5,4,4,4,3,2,1,1,1,1,1,1,1]
print(len(lats), len(lons), len(data))

# Create a meshgrid for interpolation
xi = np.linspace(-180, 180, 100)
yi = np.linspace(-90, 90, 100)
xi, yi = np.meshgrid(xi, yi)

# Different functions for the spherical interpolation:
```

```python
# 'linear': Linear radial basis function
# 'multiquadric': Multiquadric radial basis function
# 'inverse': Inverse radial basis function
# 'gaussian': Gaussian radial basis function
# 'cubic': Cubic radial basis function
# 'quintic': Quintic radial basis function
# 'thin_plate': Thin-plate spline radial basis function

# Perform 2D interpolation using Rbf for spherical interpolation
rbf = Rbf(lons, lats, data, function='linear')
zi = rbf(xi, yi)

# Create a Cartopy PlateCarree projection centered at lon = 0
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=0))

# Add coastlines and gridlines
ax.coastlines()
ax.gridlines(draw_labels=True)

# Set extent of the plot
ax.set_extent([-180, 180, -90, 90], crs=ccrs.PlateCarree())

# Create color plot
plt.pcolormesh(xi, yi, zi, shading='auto', cmap='viridis')

# Create scatter plot with data as color
scatter = ax.scatter(lons, lats, c=data, cmap='viridis', s=100,\
                     edgecolors='black',linewidths=1,\
                     transform=ccrs.PlateCarree())

# Add colorbar
cbar = plt.colorbar(scatter, label='Number of physics nobel prize winners by country', ax=ax, shrink=0.7, pad = .1)
cbar.mappable.set_clim(0, 91)

# Add title
plt.title('2D Interpolation Colorplot with\nScatter Plot (PlateCarree Projection)')

plt.show()
```