Ana Vitoria Rodrigues Lima **Homework 1** AM 205
September 23, 2021

Problem 1

(a) Given a Gamma function defined as:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt \tag{1}$$

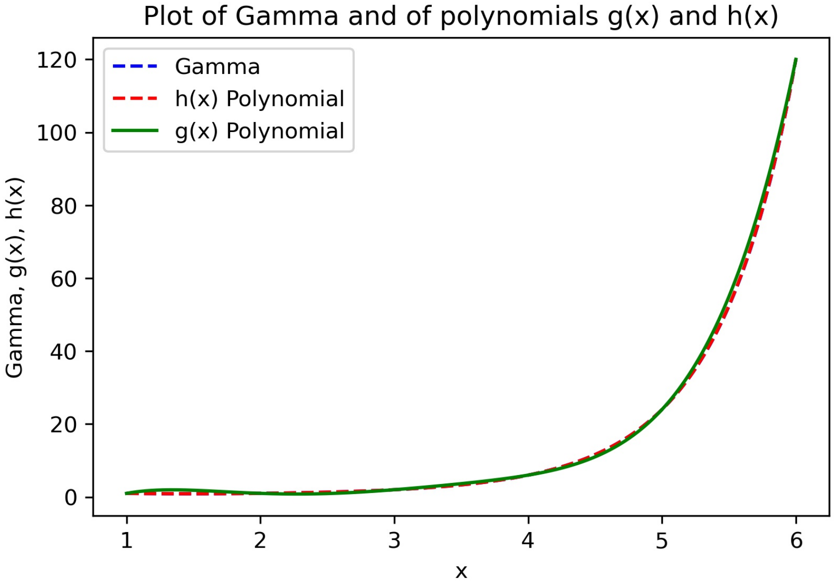
I constructed an approximation for the Gamma function by finding a g(x) polynomial of 5^{th} degree interpolant in the points given in the exercise. To find the coefficients of the polynomial interpolant of 5^{th} degree the Vandermode matrix (with the numpy python command np.vander()) has been used. Given y=Vb, the coefficients can be backsolved with b=Vy with the command np.linalg.solve(). Out of these calculations, the interpolant $g(x) = \sum_{k=0}^{5} g_k x^k$ is given by:

$$g(x) = -35 + 83.8833x^1 - 70.875x^2 + 27.75x^3 - 5.125x^4 + 0.36667x^5 \tag{2}$$

(b) In part (b) of Problem 1 I create another approximation of Gamma by first taking another step, i.e. approximating the log of its outputs. This new approximation, h(x), is a fifth order polynomial. To achieve this, I first calculate a fifth order polynomial p(x) that interpolates the points $(n, \log(\Gamma(n)))$ for $n = 1, 2, 3, 4, 5, 6$. To get h(x), the exponential of p(x) is taken. To find these coefficients, the command used has been np.linalg.solve(). This results in:

$$h(x) = e^{1.267 - 2.187x^1 - 1.101x^2 - 0.201x^3 - 0.0217x^4 + 0.0009721x^5} \tag{3}$$

(c) Plotting the three functions mentioned above: the given $\Gamma(x)$ to be approximated and its two approximations $g(x)$ and $h(x)$.

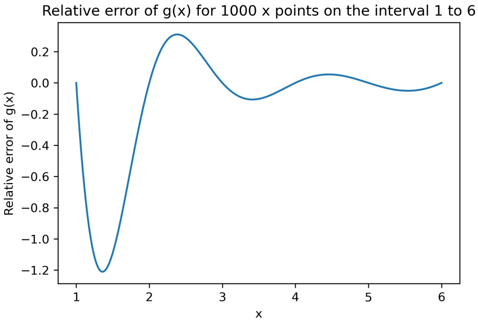


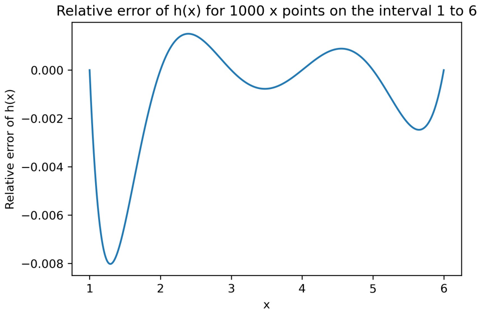
(d) To find the maximum relative error between $\Gamma(x)$ and $g(x)$ and the maximum relative error between $\Gamma(x)$ and $h(x)$ I calculate the relative error for 1000 points equally spaced in the range between 1 and 6, then I get the absolute value of this, and find the maximum of the absolute value. The max of the relative errors hence are:

$$max_{x[1,6]}|\Gamma(x) - g(x)|/\Gamma(x) = 1.210837726 \tag{4}$$

$$max_{x[1,6]}|\Gamma(x) - h(x)|/\Gamma(x) = 0.0080211437 \tag{5}$$

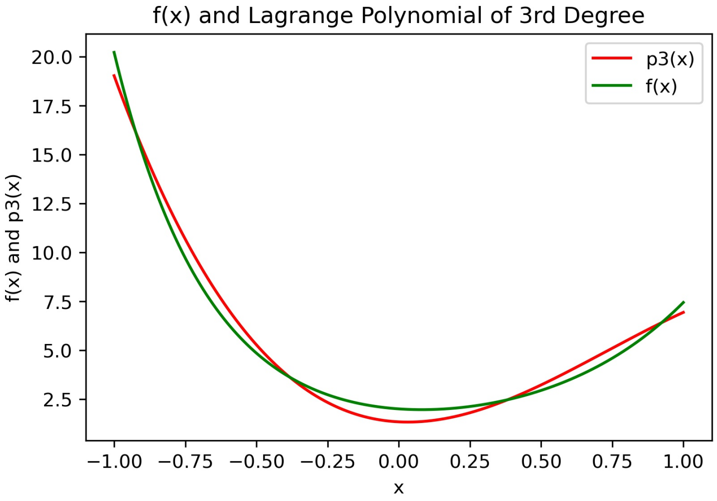
These can maximum absolute values can also be grahically seen here:





Problem 2

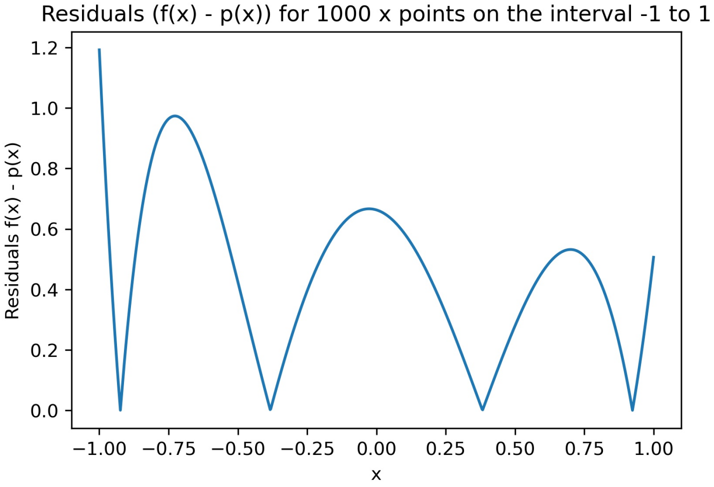
(a) Given the function $f(x) = e^{-3x} + e^{2x}$, I programmed in python a function to calculate and plot the Lagrange polynomial $p_{n-1}(x)$ of $f(x)$ at the Chebyshev points $x_j = \cos((2j-1)\pi/2n)$ for $j = 1, \ldots, n$ by following one of the coding examples provided in class (the example "lsum.py"). Then, I plotted the Langrange Polynomial $p_3(x)$ and the given function $f(x)$ for 1000 points over the range $[-1, 1]$.



(b) To calculate the infinity norm, calculated the residuals for 1000 equally spaced points over $[-1, 1]$ and selected the maximum absolute value out of these residuals. The infinity norm turns out to be:

$$||f - p_3||_\infty = max_{x[-1,1]}|f(x) - p(x)| = 1.19248863 \tag{6}$$

The erros can be visualised as:


Residuals (f(x) - p(x)) for 1000 x points on the interval -1 to 1

(c) Here follows a derivation of the upper bound of $||f - p_{n-1}(x)||_\infty$ given a function $f(x) = e^{-3x} + e^{2x}$ and given the inputs being Chebyshev points. The infinity norm is given by:

$$||f(x) - p_{n-1}(x)||_\infty = max_{x[-1,1]}|f(x) - p_{n-1}(x)| \tag{7}$$

From the lecture, we recall the error equation:

$$f(x) - p_{n-1}(x) = \frac{f^n(\theta)}{n!}(x - x_1)(x - x_2)...(x - x_n) \tag{8}$$

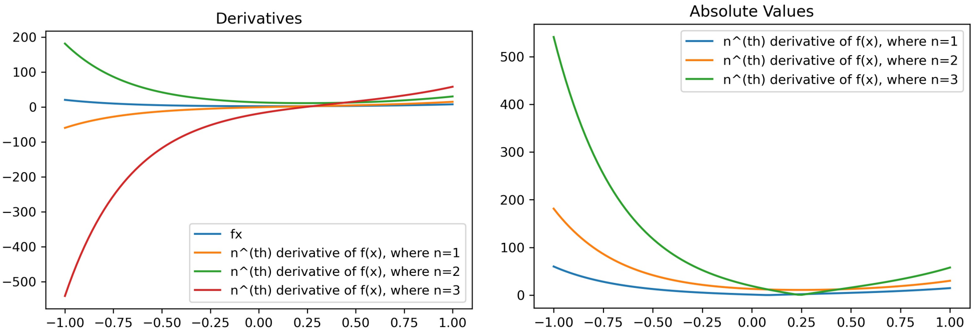
$$f(x) - p_{n-1}(x) < ||\frac{f^n(\theta)}{n!}(x - x_1)(x - x_2)...(x - x_n)|| \tag{9}$$

We can split the search for the upper bound of the error equation in (8) in two parts: firstly I try to find the maximum of the first term $\frac{f^n(\theta)}{n!}$, secondly I try to maximise the second term $(x - x_1)(x - x_2)...(x - x_n)$.

To tackle the first term maximization, by starting from its nominator, the nth derivative of $f(\theta)$ is:

$$f^n(\theta) = (-3)^n e^{-3\theta} + 2^n e^{2\theta} \tag{10}$$

Additionally, the maximum value of θ that maximises $f^n(\theta)$ is -1. This can be seen by the following graphs. Given some n values $n[1,3]$, the nth derivative of $f(\theta)$ achieves maximum absolute value where θ approaches/is at its end point -1.



Given the maximum of each these polynomials occurs at the boundary point -1, the first term can be considered maximum at:

$$||f^n(\theta)|| = (-3)^n e^{(-3)(-1)} + 2^n e^{2(-1)} \tag{11}$$

$$||f^n(\theta)|| = (-3)^n e^3 + 2^n e^{-2} \tag{12}$$

Now I turn to the second part of the equation, to the term $||(x-x_1)...(x-x_n)||$. By using Chebyshev points, we are already minimizing this maximum value to $T_n(x)/2^{n-1}$. This however still means that the maximum that this term can achieve, given that our input points are Chebyschev points, is $T_n(x)/2^{n-1}$. Hence I re-write the second term as:

$$||(x-x_1)...(x-x_n)||_\infty = ||T_n(x)/2^{n-1}|| \tag{13}$$

Chebyshev points, coming into place from a cos function (this being $cos(cos^{-1}x)$), oscillate between [-1,1]. Hence the maximum of $||T_n(x)||$ is 1. Hence I re-write the second term as $\frac{1}{2^{n-1}}$, resulting into:

$$||(x-x_1)...(x-x_n)||_\infty = 1/2^{n-1} \tag{14}$$

After having broken down our maximization problem, we see that the notation above, the error equation (8), will have its upper bound as follows:

$$f(x) - p_{n-1}(x) = \frac{f^n(\theta)}{n!}(x-x_1)(x-x_2)...(x-x_n) \tag{15}$$

$$||f(x) - p_{n-1}(x)||_\infty < ||\frac{f^n(\theta)}{n!}(x-x_1)(x-x_2)...(x-x_n)|| \tag{16}$$

$$||f - p_{n-1}(x)||_\infty < max_{\theta[-1,1]} \frac{-3^n e^{-3\theta} + 2^n e^{2\theta}}{n!} \frac{1}{2^{n-1}} \tag{17}$$

$$||f - p_{n-1}(x)||_\infty < max_{\theta[-1,1]} \frac{-3^n e^{-3(-1)} + 2^n e^{2(-1)}}{n!} \frac{1}{2^{n-1}} \tag{18}$$

$$||f - p_{n-1}(x)||_\infty < max_{\theta[-1,1]} \frac{-3^n e^{+3} + 2^n e^{-2}}{n!} \frac{1}{2^{n-1}} \tag{19}$$

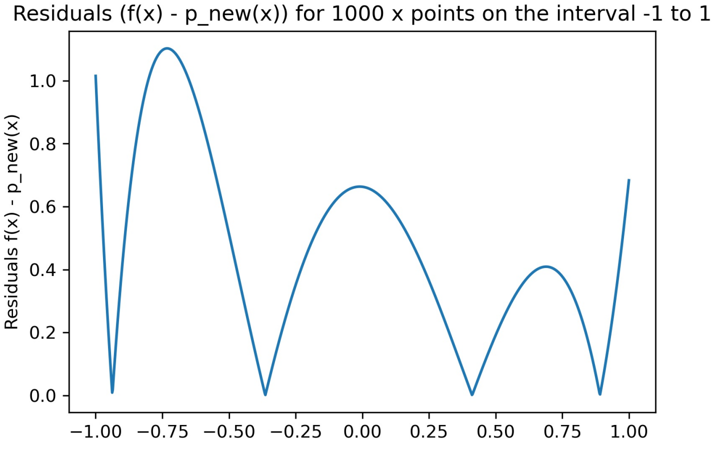
(d) Here I have tried to calculate a cubic polynomial p_3^+ of 3^{rd} degree such that its infinity norm $||f - p_3^+||_\infty$is smaller than the one calculated in 2.b for the polynomial p_3. To do this, I calculated the coefficients of a Lagrange polynomial as a cubic polynomial plotted over a 1000 equally spaced points, to then manipulate this new cubic polynomial, up to achieving $||f - p_3||_\infty < ||f - p_3^+||_\infty$. The coefficients of p_3 were [1.34 -0.75 11.643 -5.325]. These resulted in the following infinity norm:

$$max_{x[-1,1]}|f - p_3| = 1.19248863 \tag{20}$$

Whereas with a twicked coefficient, namely coefficients coefficients of p_3^+ = [1.34 -0.9 11.643 -5.325], i.e. with the polynomial p_3^+ = 1.34 - 0.9x + 11.643 x^2 - 5.325 x^3, the infinity norm is:

$$max_{x[-1,1]}|f - p_3^+| = 1.10291798 \tag{21}$$

This can also be seen visually from this plot:



Problem 3

(a) Let's find 2x2 invertible matrices B and C such that k(B+C) < k(B) + k(C).

$$B = \begin{bmatrix} 6 & 0 \\ 0 & 1 \end{bmatrix}, B^{-1} = \begin{bmatrix} 1/6 & 0 \\ 0 & 1 \end{bmatrix}$$

$$k(B) = ||B||||B^{-1}|| = 6 * 1 = 6$$

$$C = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, C^{-1} = \begin{bmatrix} 1/10 & 0 \\ 0 & 1/10 \end{bmatrix}$$

$$k(C) = ||C||||C^{-1}|| = 10 * (1/10) = 1$$

$$(B + C) = \begin{bmatrix} 16 & 0 \\ 0 & 11 \end{bmatrix}, (B + C)^{-1} = \begin{bmatrix} 1/16 & 0 \\ 0 & 1/11 \end{bmatrix}$$

$$k(B + C) = ||B + C||||(B + C)^{-1}|| = 16 * 1/11 = 1.4545$$

$$k(B + C) < k(B) + k(C) = 1.4545 < 6 + 1 = 1.45 < 7$$

(b) Let's find 2x2 invertible matrices B and C such that k(B+C) > k(B) + k(C).

$$B = \begin{bmatrix} 6 & 0 \\ 0 & 1 \end{bmatrix}, B^{-1} = \begin{bmatrix} 1/6 & 0 \\ 0 & 1 \end{bmatrix}$$

$$k(B) = ||B||||B^{-1}|| = 6 * 1 = 6$$

$$C = \begin{bmatrix} -2 & 0 \\ 0 & -2/3 \end{bmatrix}, C^{-1} = \begin{bmatrix} -1/2 & 0 \\ 0 & -3/2 \end{bmatrix}$$

$$k(C) = ||C||||C^{-1}|| = 2 * (3/2) = 3$$

$$(B + C) = \begin{bmatrix} 4 & 0 \\ 0 & 1/3 \end{bmatrix}, (B + C)^{-1} = \begin{bmatrix} 1/4 & 0 \\ 0 & 3 \end{bmatrix}$$

$$k(B + C) = ||B + C||||(B + C)^{-1}|| = 4 * 3 = 12$$

$$k(B + C) > k(B) + k(C) = 12 > 6 + 3 = 12 > 9$$

Problem 4

(a) Given 4 spaces, I want to find 4 cubic polynomials that can piece-wise interpolate a cubic periodic polynomial passing through the given points (t,x), these being $(t, x) = (0, 0), (1, 1), (2, 0), (3, -1)$. To solve this, it means I need to find 16 coefficients, and to do that I need 16 constraints. These are given by 8 derivative conditions (so that it's a continuous curve) and 8 points constraints conditions. The constraints points conditions are:

$$S_1(0) = 0, S_1(1) = 1$$

$$S_2(1) = 1, S_2(2) = 0$$

$$S_3(2) = 0, S_3(3) = -1$$

$$S_4(3) = -1, S_4(4) = 0$$

The constraints from the derivative conditions are:
The 8 derivative conditions:

$$S_1(0)' = S_4(4)' S_1(0)'' = S_4(4)''$$
$$S_1(1)' = S_2(1)' S_1(1)'' = S_2(1)''$$
$$S_2(2)' = S_3(2)' S_2(2)'' = S_3(2)''$$
$$S_3(3)' = S_4(3)' S_3(3)'' = S_4(3)''$$

Using the CubicSpline function in python, the Sx cubic spline comes into place piecewise for 4 spaces (t from 0 to 1, t from 1 to 2, and so on), the 4 polynomials Sx_i are:

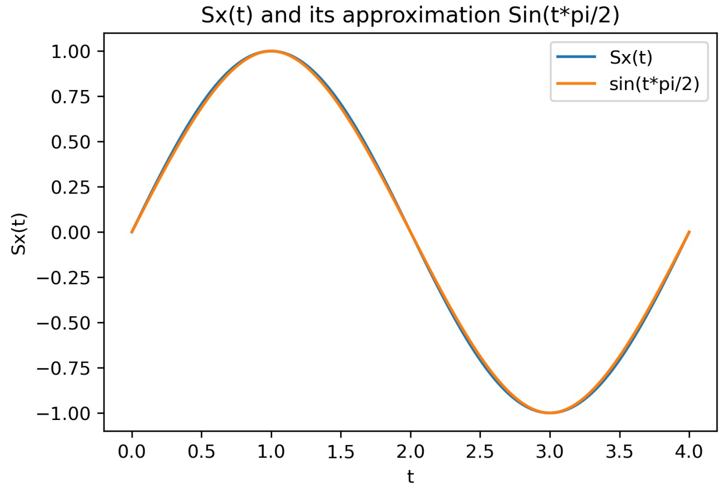
$t_1 = [0, 1] : Sx_1 = 1.5t_1 - 0.5t_1^3$

$t_2 = [1, 2] : Sx_2 = 1.0 - 6.846e^{-17}t_2 - 1.5t_2^2 + 0.5t_2^3$

$t_3 = [2, 3] : Sx_3 = -1.49999t_3 - 4.441e^{-16}t_3^2 + 0.5(t_3)^3$

$t_4 = [3, 4] : Sx_4 = -1.0 - 6.476e^{-17}t_4 + 1.5t_4^2 - 0.5t_4^3$

(b) Here I am plotting $sin(t\pi/2)$ with the cubic spline $S_x(t)$ for 1000 points in t = [0,4].



(c) Similarly to Sx, to find Sy I find 4 cubic polynomials coming into place from 16 constraints. The constraint points this time being $(t, x) = (0, 1)$, $(1, 0)$, $(2, -1)$, and $(3, 0)$. The Sy comes into places from Sy_i for 4 spaces, these are:

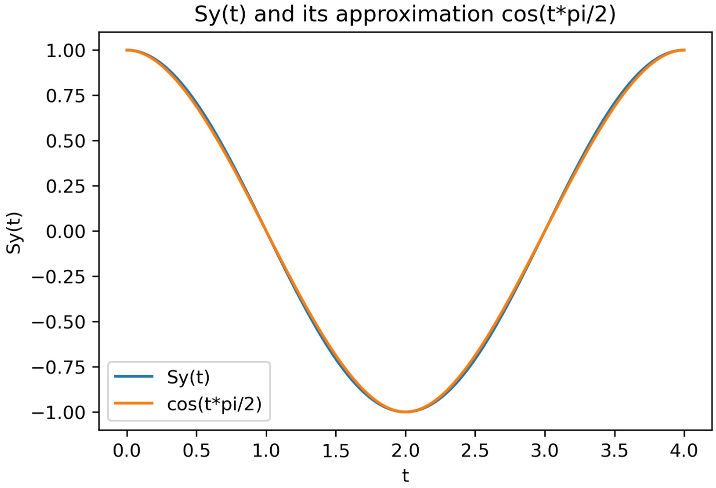
$t_1 = [0, 1] : Sy_1 = 1 + 1.11e^{-16}t_1 - 1.5t_1^2 + 0.5t_1^3$

$$t_2 = [1,2] : Sy_2 = 0.0 - 1.5t_2 + 2.22044e^{-16}t_2^2 + 0.5t_2^3$$

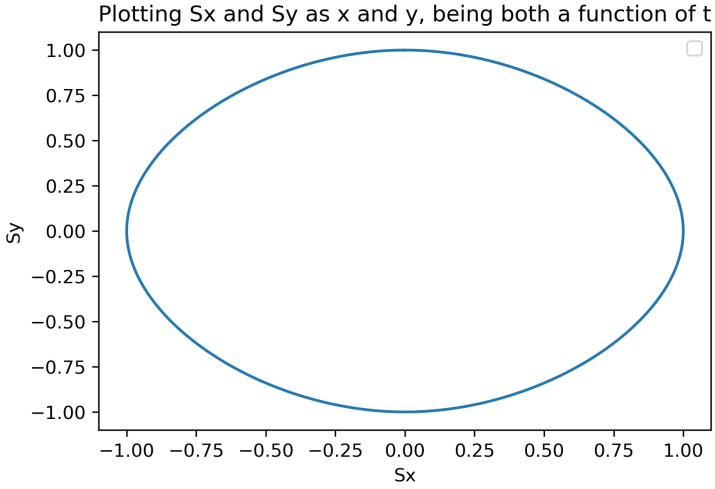
$$t_3 = [2,3] : Sy_3 = -1 + 1.110223e^{-16}t_3 + 1.5t_3^2 - 0.5(t_3)^3$$

$$t_4 = [3,4] : Sy_4 = 0.0 + 1.49999999t_4 + 2.220446e^{-16}t_4^2 - 0.5t_4^3$$

The plot of the polynomial $Sy_(t)$ against $cos(t\pi/2)$ follows:



(d) Being that both Sx and Sy depend on t, I can both plot them against each other. After this, I use scipy.integrate.simps to find the area under the curve. Given that the radius is 1, as given by the exercise, the area already approximates our π. The approximation found has been of 3.049999.



Problem 5

(a) In problem 5, the aim is to re-construct the image 'regular.png' by using a linear regression whose inputs are going to be three low light pictures: low1, low2, low3. These three low light pictures were taken in such a way for these to have a predominant channel. In any given image, each pixel has three channels: a red channel, a green channel and a blue channel. So that a pixel could be seen as a 3-elements array.

The aim is to train a linear regression, so that this can later re-construct the pixels of the real 'regular.png' image by putting together those three channels.
To do so, I train three regressions, one that will have the coefficients for the red channel of pixels, another regression that will have the coefficients for the green channel of the coefficients, and lastly another regression that will have the coefficients for the blue channel of coefficients.

The first step, is to read all our images with io.imread() and divide this by 255, so that the pixel can store a value between 0 and 1.
The shape of the regular photo is a (400, 300, 3) matrix, with red pixels, green pixels, and blue pixels each as a layer in the image as the third dimension. This is also the shape of the low1, low2 and low3 images. However, to use these, I reshape the low1, low2 and low3 image to have each a matrix size (120000, 1).

Given that I need a dataset to fit my linear regression (done with the command LinearRegression()), I concatenate these 3 low images with axis=1, i.e. nexto to each other. This matrix will have a shape of (120 000, 9).

To estimate b, I need to fit the model to a matrix X but also to some outputs y. I chose these outputs y to be the three different channels of each pixel in the real 'regular.png' image. By doing this, I have 3 different linear regression models, which will find the red channel of a pixel of the real image, the green channel of a pixel of the real image, and the blue channel of a pixel in the real image.

Additionally, the constant vector is found with the command .intercept_ in python.
My three linear regression models are red_linreg, green_linreg and blue_linreg.

The coefficients of the red_linreg model are [0.60008372 -0.83676669 -2.30778801 1.98194378 0.93751476 -1.49131774 -0.31465748 0.18149806 0.32879883].

The coefficients of the green_linreg model are [-0.35171075 0.04388672 -1.50469715 1.20186615 2.04424222 -2.0169704 -0.47451353 0.55429676 0.30397059].

The coefficients of the blue_linreg model are [-0.38399305 -0.43968442 1.00330556 0.86871427 -0.27394525 -0.67605163 -0.13004278 0.4885107 1.02908663].

The intercepts are [0.06216703] [0.04630345] [-0.00627709]]

The reconstructed image will come into place from:

$$p_k^A = F^B p_k^B + F^C p_k^C + F^D p_k^D + p_{\text{const}}$$

Where the F^B, F^C, F^D, are:

$$Fb = \begin{bmatrix} 0.6 & -0.84 & -2.31 \\ -0.35 & 0.044 & -1.5 \\ -0.38 & -0.434 & 1.00 \end{bmatrix}$$

$$Fc = \begin{bmatrix} 1.98 & 0.94 & -1.49 & \\ 1.2 & 2.04 & -2.02 & \\ 0.87 & -0.27 & -0.68 & \end{bmatrix}$$

$$Fd = \begin{bmatrix} -0.31 & 0.18 & 0.33 \\ -0.47 & 0.55 & 0.304 \\ -0.13 & 0.49 & 1.03] \end{bmatrix}$$

The intercepts are: $p_{const} = \begin{bmatrix} 0.062 \\ 0.046 \\ -0.0062 \end{bmatrix}$

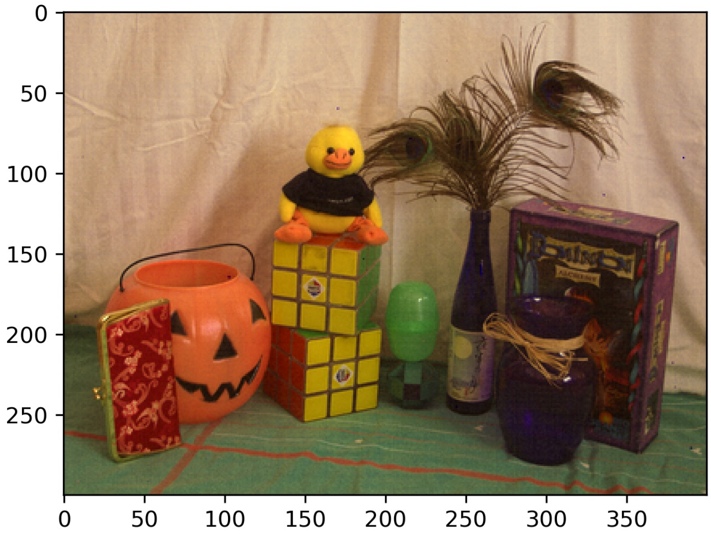
I use these three models with the python function .predict () three times with the input X (the aforementioned matrix built from low1, low2 and low3 with shape of 120000x9) and I get the predicted red, green and blue channels.

After I did this, I concatenate these predicted RGB channels and I get an output matrix of 120 000 rows and 3 columns, these are all my pixels of the regular image with the re-constructed three colour channels. However, to be able to see this as an image, I reshape this to (300, 400, 3).

S minimized is:

$$S = \frac{1}{MN} \sum_{k=0}^{MN-1} \|F^B p_k^B + F^C p_k^C + F^D p_k^D + p_{\text{const}} - p_k^A\|_2^2. = 0.0040495 \tag{22}$$

Here my re-constructed image:

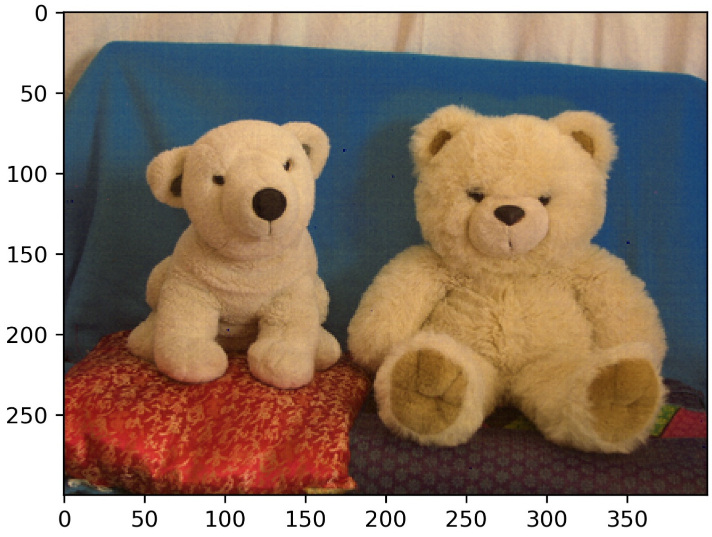


Here is the original 'regular.png' image:

(b) After having trained the model in (a), here I use the aforementioned three linear models to re-create the bears image. I am using the aforementioned coefficients but this time using as input for the big input X matrix (120 000,9) the low1, low2 and low3 images of the bears. The mean squared error of this calculations is:

$$T = \frac{1}{MN} \sum_{k=0}^{MN-1} ||p_k^A - p_k^{A*}|| \ = 0.0054 \tag{23}$$

Here is the re-constructed image of the bears Orsetto Grande and PB the Eager:



Here is the original image of the bears Orsetto Grande and PB the Eager: