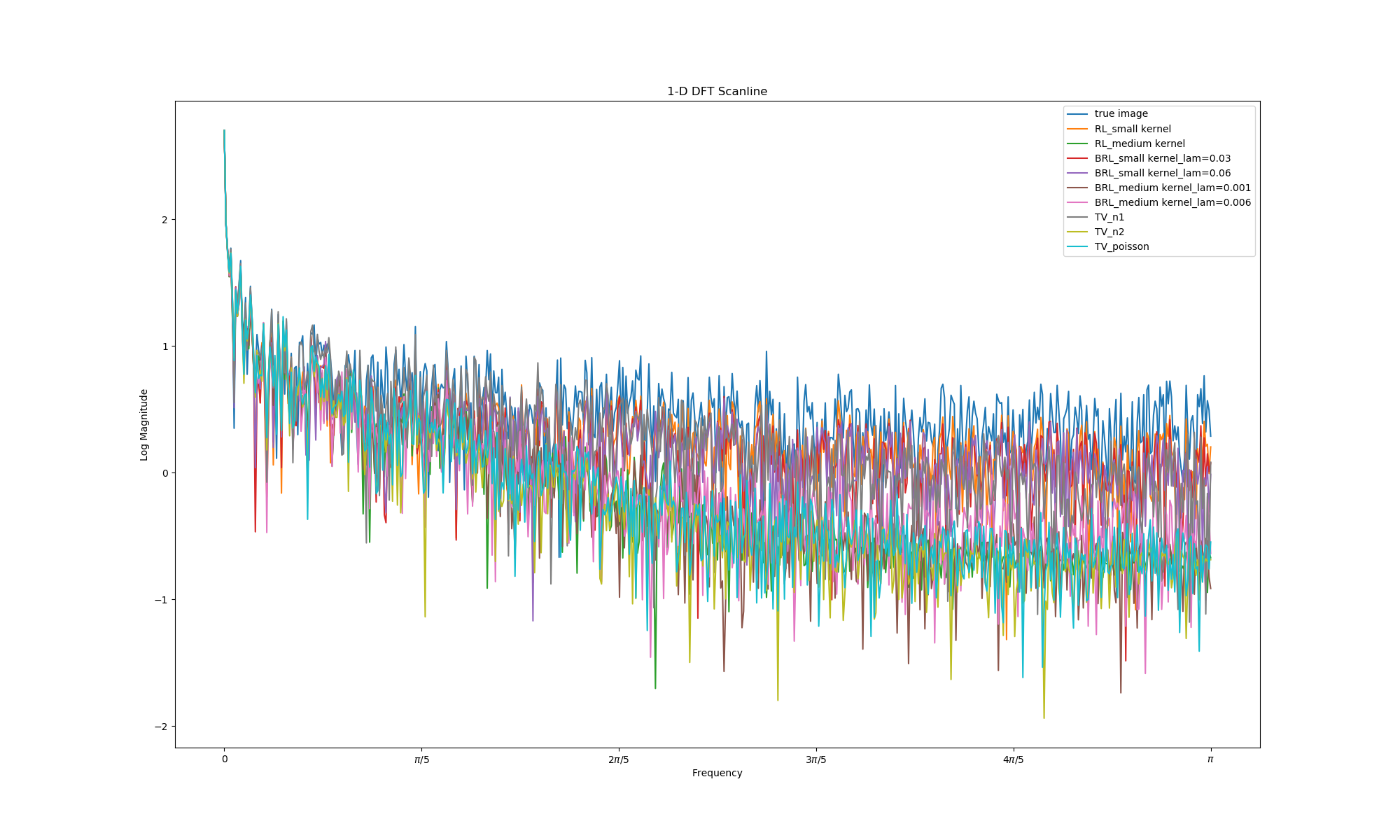
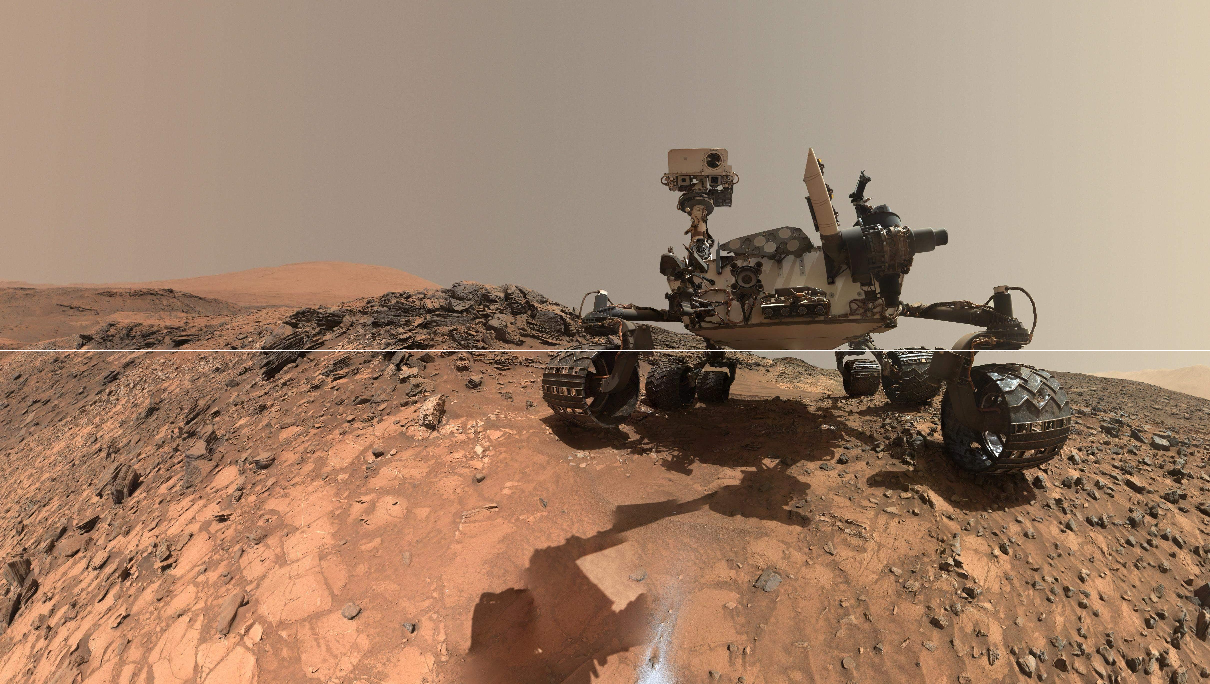
* Experiment – a
  + Try to plot a same figure for the deblurred results and a non-blurred image data/curiosity.png





註:上圖的水平白線處為我選的scanline

* + How do you plot the figure? Do you apply any additional steps? Why?

首先將Implementation中所有deblur method產出的圖與non-blurred image都用np array儲存成三維陣列，並從圖中選定一條水平線當作scanline，我選定的水平線位在shape[0]為350的位置。並將剛剛上面所有轉成np array的圖片之該水平線位置取一維DFT，一維DFT步驟如下：

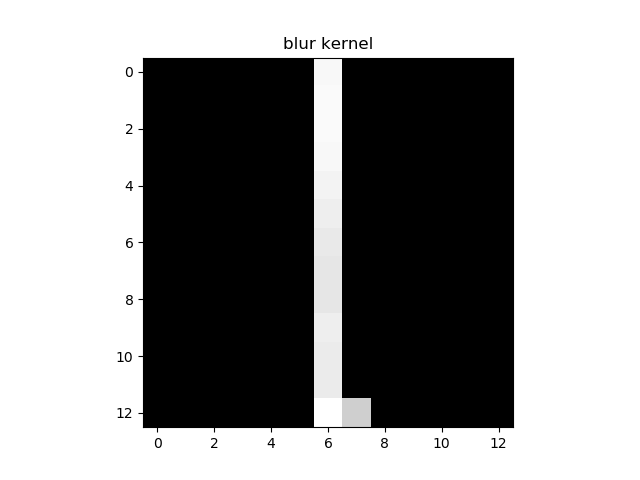
1. 水平線位在shape[0]為350的位置之pixel value值，先對其3個channel取平均
2. 取平均後的array即為一維陣列，每個element都除以255，做normalization，將value壓在0到1之間
3. 接著用np.fft.fft()做one-dimensional discrete Fourier Transform
4. 再將np.fft.fft()之output(也是一維陣列)的每個element取log(以10為底)，需特別注意的是，因為log的定義域為大於零的數，故須將np.fft.fft()之output取絕對值再取log。最後產出的一維陣列即為一維DFT的結果(DFT\_output)。

而作業說明中所附圖之X軸frequency domain是0到pi，因為是一半的週期，所以這邊我們將一維的DFT結果取前半當作Y軸，X軸則是0到pi的等差數列(總共有(len(DFT\_output)/2)個數字)，並以plt.plot作圖即可得到某一張圖的1-D DFT scanline。以此類推總共有十張圖都重複上述步驟即可。

* + How this figure can help you to explain the result?

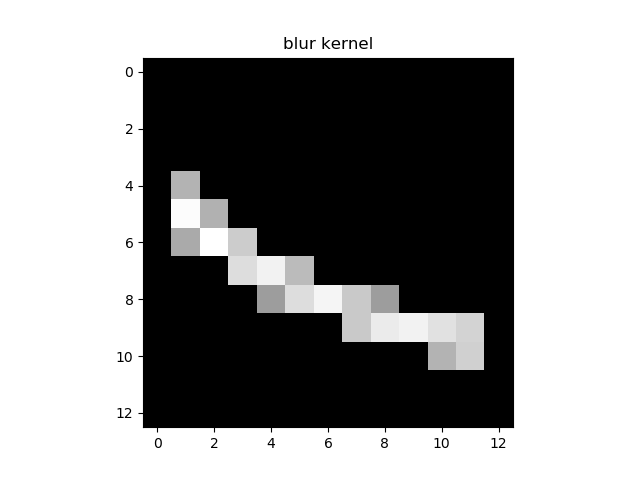
我們可以看出，沒有晃動的true\_image其Log Magnitude最大，然後TV\_poisson最小，且 deblur效果越好的方法，其Log Magnitude越大，越逼近true\_image的Log Magnitude，並且可由該折線圖清楚發現total varitation的作法，不論是norm-1或norm-2或poisson其deblur效果都輸給RL或BRL，因為total varitation method的Log Magnitude幾乎都是墊底，而BRL因為多了spational & range penalty，預期BRL的Log Magnitude應該會比RL都高一個檔次，但從折線圖中發現也不盡然是如此。

* Experiment – b
  + blurred images with straight kernel (程式碼位置：/code/ Experiments\_b\_straight\_kernel.py)
    - straight kernel (13\*13)



|  |  |
| --- | --- |
| my\_image\_straight.png | my\_deblur\_straight.png |

* blurred images with curl kernel(程式碼位置：/code/ Experiments\_b\_curl\_kernel.py)
  + curl kernel(13\*13)



|  |  |
| --- | --- |
| my\_image\_curl.png | my\_deblur\_curl.png |

* + how you choose your parameters

Straight kernel與curl kernel的BRL參數我都設為下表:

|  |  |
| --- | --- |
| max\_iter\_BRL | 25 |
| rk | 6 |
| Sigma\_r | 50.0/255/255 |
| lamb\_da | 0.03/255 |
| to\_linear | ‘False’ |
| r\_omega | 0.5\*rk |

試過非常多組參數組合，幾乎都與上表所產出結果雷同。

* + some extra preprocess on your blur kernels

blur kernel的作法如下：

1. 用小畫家剪出蛋黃哥左上角的方框
2. 用opencv resize成downsample成13\*13的圖片，並以灰階儲存
3. 再對這13\*13個pixels做一個threshold判斷，若沒有超過某個threshold，都將其設為0。
4. 因為curl kernel做resize後在邊界有些許雜訊，故curl kernel有額外做一個判斷，讓13\*13的邊界處的值都設成0。

註:步驟3與步驟4的目的都是希望讓kernel的形狀很接近straight與curl，並且使其清晰易見。

* + compare the the deblur effect for the two kernels

Deblur效果沒有很好的原因，推估是因為我將照片拍得太晃了。不過還是可以看出deblur成效，像是圖下方的工作列有變比較清楚，且蛋黃哥圖片左上角的黑色方框區域內kernel線，可看出不論是straight或是curl，雖然沒有變回一個點，但都從完整的一條線變成三個分段，可見其deblur成效。而不論是straight kernel或是curl kernel，用BRL做完都有一點ringing effect。

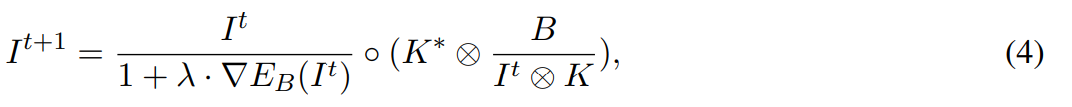
* Free study – Problem 1.
  + Compare the RL and BRL results of Curiosity-small and Curiosity-medium
    - Assumption 1

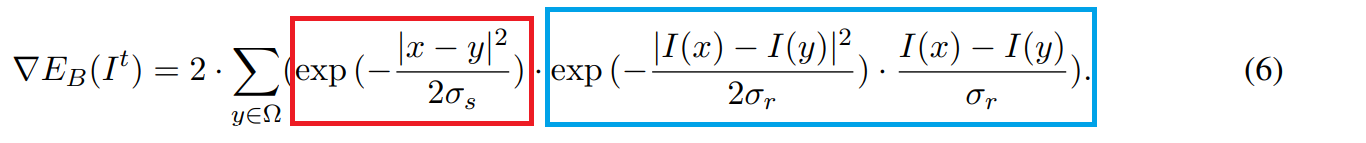
因為沒有任何的penalty限制，更新時容易造成圖像物體邊緣處(pixel value差距較大的部分)能量傳遞擴散，故RL會有ringing effect。且當kernel size越大，能量可以傳遞越遠，ringing effect的波紋就會越大且越明顯。

* + - Assumption 2

BRL加入了spatial penalty(下式紅框處)與range penalty(下式藍框處)。尤其是range penalty，在圖像的物體邊緣I(x)與I(y)差距較大，此時range penalty變大，造成變大，此時的更新就較不依賴當前的，讓ringing effect較RL不明顯。

* + - Assumption 3





當BRL之rk值越大，sigma\_s值就會越大，此時spatial penalty(上式紅框處)變小，會跟著變小，而再根據上面式(4)，變小時，表示某次的更新較依賴當前的，所以在消除晃動表現上，會較不明顯。

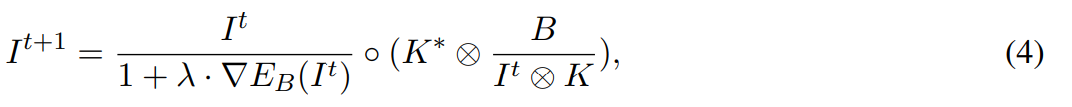
* + - Justification

為了驗證我的假設，我做了如下的對照：(取自Implementation)

|  |  |
| --- | --- |
| RL, Curiosity-small, iter=25 | BRL, Curiosity-small, iter=25, rk=6, sigma\_r=50, lamb\_da=0.03 |
| RL, Curiosity-medium, iter=55 | BRL, Curiosity-medium, iter=55, rk=12, sigma\_r=25, lamb\_da=0.001 |

可以發現RL都會存在ringing effect，且Curiosity-medium(kernel size=25\*25)比Curiosity-small (kernel size=13\*13)的ringing effect更明顯，這邊就驗證了我上面的Assumption 1(上表的左上圖跟左下圖比較)。而BRL因為加入spatial penalty與range penalty，讓ringing effect較RL不明顯(上表的左圖跟右圖比較，驗證Assumption 2)，但是會有一層糊糊的油畫感，且rk值越高這種模糊感越明顯(上表的右上圖跟右下圖比較)，這邊就驗證了我上面的Assumption 3，因為rk值高造成每次的更新梯度低，解決方法之一是讓更新次數變得更多，就有機會消除模糊感。

* + Compare the BRL results of Curiosity-medium with different parameters.
    - Compare lamb\_da (其他參數都固定，iteration=25, sigma\_r=50/255/255, rk=6)
      * Assumption



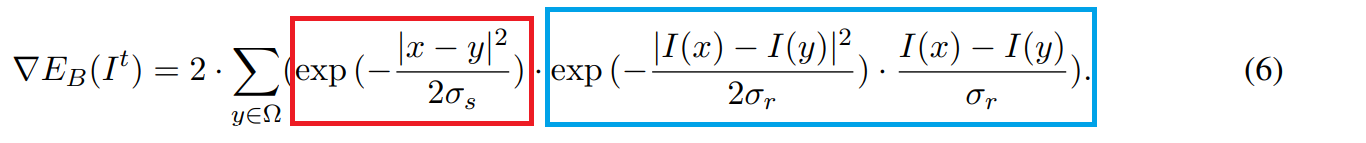
當lamb\_da值越大，分母越大，使的spatial penalty與range penalty的權重提高，所以畫面更加smooth。並可由上式發現更新時受影響也會越小，且更新多半受上面4式後面黑框處所控制，所以能夠更有效消除ringing effect。

* + - * Justification

|  |  |
| --- | --- |
| lamb\_da=0.03/255 | lamb\_da=0.3/255 (圖片位置:/my\_RL\_BRL\_result/Free\_study\_p1/ BRL\_s\_iter25\_rk6\_si50.00\_lam0.300.png) |

為了驗證我的assumption，我將其他參數固定(iteration=25, sigma\_r=50/255/255, rk=6)，只改動lamb\_da值，當lamb\_da放大十倍的情況下，可以看到圖像更加smooth，並且有一點油畫不真實感。

* + - Compare sigma\_r (其他參數都固定，lamb\_da=0.03/255, iteration=25, rk=6)
      * Assumption



由上式可發現，當sigma\_r越大，後面藍框處的range penalty就會越小，當圖像的物體邊緣I(x)與I(y)差距不變的情況下，因sigma\_r在分母，造成變小，此時的更新就較依賴當前的，故推論ringing effect會較明顯。

* + - * Justification

|  |  |
| --- | --- |
| sigma\_r = 50.0/255/255 | sigma\_r = 100.0/255/255 (圖片位置:/my\_RL\_BRL\_result/Free\_study\_p1/ BRL\_s\_iter25\_rk6\_si100.00\_lam0.030.png) |

為了證明我的推論，我將sigma\_r放大兩倍變成100.0/255/255，其他參數固定(lamb\_da=0.03/255, iteration=25, rk=6)，可發現在右圖好奇號的物體邊緣處，有些微的ringing effect產生。由此可驗證我的假設。

* Free study - Problem 2. Compare the BRL results of Curiosity-medium with different boundary condition.
  + Assumption

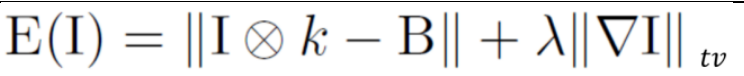
Implementation部分採用sysmetric padding的方式，而這邊我想嘗試採用zero padding來看看效果差異，因為zero padding是將突出原圖的部分pixel value都補0，而這樣可能造成圖像邊角處的pixel value差距會很大，可能比圖片中物體的邊緣pixel value差距還大，所以推測在圖像邊角處的ringing effect應該會相當明顯。

* + Justification

|  |  |
| --- | --- |
| Medium kernel, lamb\_da=0.001/255, iter=55, sigma\_r=25/255/255, rk=12, sysmetric padding | Medium kernel, lamb\_da=0.001/255, iter=55, sigma\_r=25/255/255, rk=12, zero padding(圖片位置: my\_RL\_BRL\_result/ Free\_study\_p2/ BRL\_s\_iter55\_rk12\_si25.00\_lam0.001zero\_padding.png) |

為了驗證我的假設，上表右圖我在其他參數不變的情況下，實作zero padding，明顯可見圖像邊角處有很明顯的ringing effect，甚至比圖像物體的邊緣處所產生的ringing effect還要明顯許多。而sysmetric padding的作法讓圖像邊角處的pixel value值很接近，使的BRL做完時，圖像邊角不太會有ringing effect情況產生。

* Free study – Problem 3. Compare some different λ on TVL1 deconvolution.
  + Assumption



如上式所示，λ在TVL1 deconvolution用來控制照片total variation的gradient之參數，當λ越大，相同的gradient下所造成的影響越大，在同樣的update iteration次數下，其晃動振波感會比較少。

* + Justification

|  |  |
| --- | --- |
| lamb\_da = 0.01, iter=1000 | lamb\_da = 0.1, iter=1000(圖片位置: my\_RL\_BRL\_result/ Free\_study\_p3/ deblur\_edgetaper\_norm1\_lam0.1.png) |

為了驗證我的推論，我用TVL1.py跑了兩種測試，上表右圖的λ是左圖的10倍，可發現振波晃動感的確隨著λ變大而有所消失，但是在同樣的update次數下，因為λ變大造成每次更新的梯度較大，會有一點失真感，某些細微處(如好奇號的胎紋)變得有點模糊。

* Free study – Problem 4. Compare the results of TVL1, TVL2 and TVpoisson.
  + Assumption

TVL2因為採用L2 norm，根據gradient的更新會比TVL1的L1 norm更加顯著，所以TVL2的效果類似將TVL1的λ值調很大，消除了振波感，但是畫面會有點失真。而TVPoisson是將input blurred image當作Poisson noise來做處理，而我們的task是deblur而不是denoise，所以推估用TVpoisson做deblur的task效果不會太好。

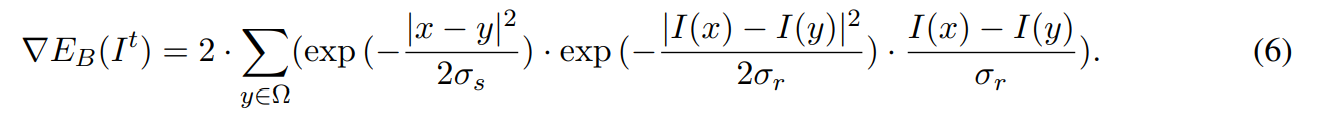
* + Justification

|  |  |
| --- | --- |
| TVL1  test\_solver = 'pc'  lam\_da=0.01  eps=0.001 |  |
| TVL2  test\_solver = 'pc'  lam\_da=0.01  eps=0.001 |  |
| TVpoisson  test\_solver = 'pc'  lam\_da=0.01  eps=0.001 |  |

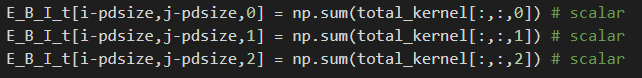
可發現在其他參數都不變的情況下，TVL1做出來效果最佳(圖像較細緻)，而TVL2就如同我所說的類似於將TVL1的λ值調很大的效果(Free study – Problem 3)，而TVpoisson的效果在這邊類似於TVL2，都帶模糊感，並且也有做到deblur的效果，這邊跟我原先假設相違背，我原本以為TVpoisson主要是用來消除poisson雜訊的，所以覺得在deblur表現上不會太好。

* Free study – Problem 5. Try to speed up BRL and report the execution time difference
  + Assumption 1

因為BRL function內有大量的矩陣相關運算，而且需要三個channel(RGB)分開計算，在運算上會花費不少時間。而我是對式6做優化，式6的數學式如下:



我原本程式碼的實作方式是將三個channel分開並且依序加總，如下:



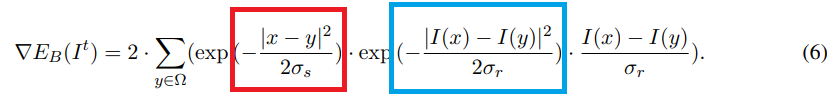
而我認為這會花費不少時間，可改成直接用一行numpy的api做平行運算，如下:



一樣都是分別對三個channel做加總，但是優化後的寫法可以同時對三個channel分開加總，我想這樣會讓運行效率更好。

* + Assumption 2

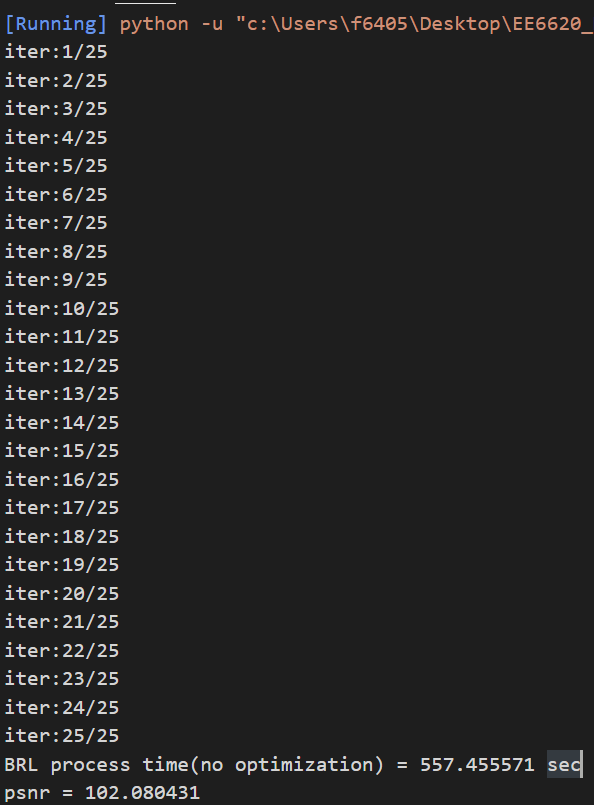
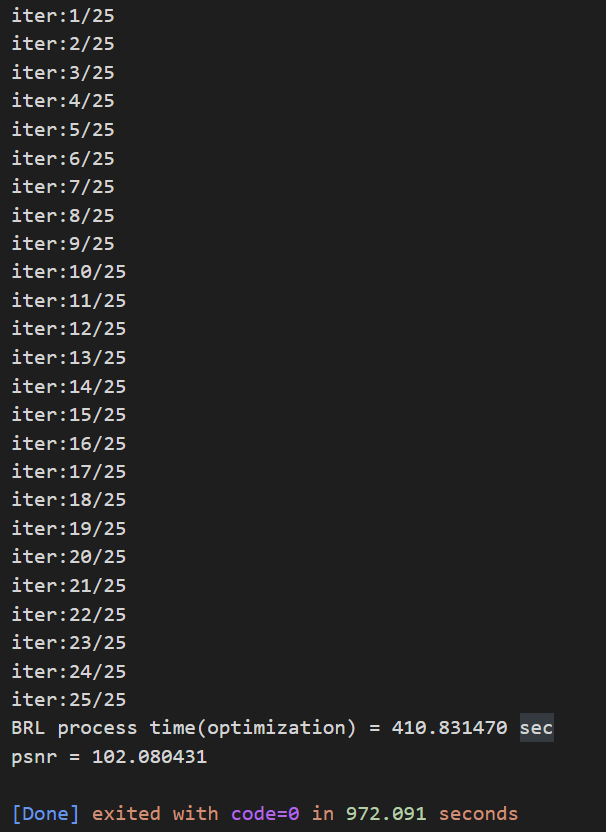
將式6的數學式子(如下圖)



根據exp性質，將紅框與藍框處先做加總再取exp，應該會比分別取exp再做相乘的效率還高，因為相加的效率比相乘還要快，並且優化後的寫法少做一次exp。

* + Justification

為了驗證我上述的兩個Assumption有讓執行效率提升，且執行出來結果一樣，我在code/Free\_study\_p5.py做實驗，分別有BRL\_no\_optimization()與BRL\_optimization()兩個function，而程式執行順序是先做BRL\_no\_optimization()再做BRL\_optimization()，兩個function傳入參數都一樣，rk=6, sigma\_r=50.0/255/255, lamb\_da=0.03/255, max\_iter\_BRL=25，並且都是對curiosity\_small.png做deblur。做完後都與reference answer算一次psnr，確認兩個優化前後的輸出結果一致。結果如下：

上面左圖是執行BRL\_no\_optimization()，上面右圖是執行BRL\_optimization()，可發現與reference answer比較psnr都一樣的情況下，優化後的執行時間比優化前快了約147秒，可見差異之顯著。