基于压缩感知的TV滤波器降噪

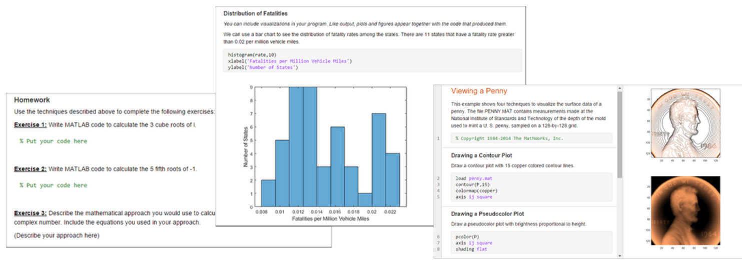
作者: 何一林 ilin12, LPS of ECNU

Email: yilinoptics@outlook.com

时间: 2020/11/03

Matlab 实时脚本mlx

实时脚本是在一个称为实时编辑器的交互式环境中同时包含代码、输出和格式化文本的程序文件。在实时脚本中，您可以编写代码并查看生成的输出和图形以及相应的源代码。添加格式化文本、图像、超链接和方程，以创建可与其他人共享的交互式记叙脚本。



详细可在help浏览器中搜索实时脚本help

图像操作

我们在图像处理过程中主要使用矩阵的方法来对图像进行操作时，通常有两种操作方式：　一，将图像进行向量化，然后按照线性代数中描述的矩阵操作方式来进行操作；二，通过对图像进行二维卷积或者三维卷积方式进行操作。

原图

```
im = double(imread('cameraman.tif'));
imshow(im, []);
title('im');
```

im

```
size(im)
```

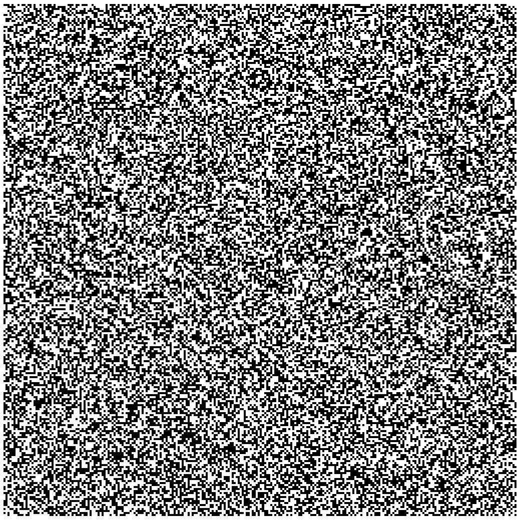
```
ans = 1×2
   256   256
```

二值编码

```
rand(4,4)
```

```
ans = 4×4
    0.8147    0.6324    0.9575    0.9572
    0.9058    0.0975    0.9649    0.4854
    0.1270    0.2785    0.1576    0.8003
    0.9134    0.5469    0.9706    0.1419
```

```
code = rand(size(im));
thold = 0.5;
code(code<thold) = 0;
code(code>=thold) = 1;
imshow(code, []);
title('code');
```




```
[height, width] = size(code)
```

```
height = 256
width = 256
```

矩阵操作

图像向量化

```
var_im = im(:);
size(var_im)
```

```
ans = 1×2
       65536           1
```

生成编码算子A

稀疏矩阵

由于内存的限制，我们需要利用matlab里面自带的稀疏矩阵这种特殊的数据结构对算子进行保存. 稀疏矩阵是一种三元组(triplet)结构, 也称为coordinate格式(coo), 它包括三个一维数组: 一个按任意顺序排列的非零矩阵元素值数组, 一个非零元所在行编号的整型数组, 以及一个非零元所在列编号的整型数组.

稀疏矩阵的生成

定义一个简单稀疏形式的对角矩阵

```
sparse(1:4, 1:4, 1:4)
```

```
ans =
   (1,1)        1
   (2,2)        2
   (3,3)        3
   (4,4)        4
```

```
full(sparse(1:4, 1:4, 1:4))
```

```
ans = 4×4
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
```

```
test_full = eye(10000);
whos test_full
```

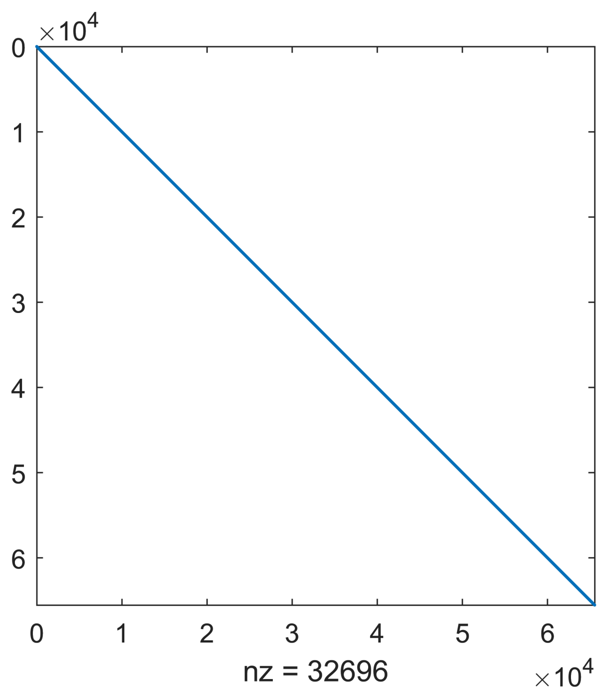
```
  Name               Size                    Bytes  Class     Attributes

  test_full      10000x10000            800000000  double
```

```
test_sparse = sparse(test_full);
whos test_sparse
```

```
  Name                 Size                 Bytes  Class     Attributes

  test_sparse      10000x10000            240008  double    sparse
```

生成编码算子的稀疏矩阵**optA**

```
var_code = code(:);
len_code = length(var_code);
optA = spdiags(var_code, 0, len_code, len_code);
spy(optA);
```




```
density = nnz(optA)/numel(optA)
```

```
density = 7.6126e-06
```

```
sparseity = 1-density
```

```
sparseity = 1.0000
```

```
[i, j] = find(optA);
half_bandwidth = max(abs(i-j))
```

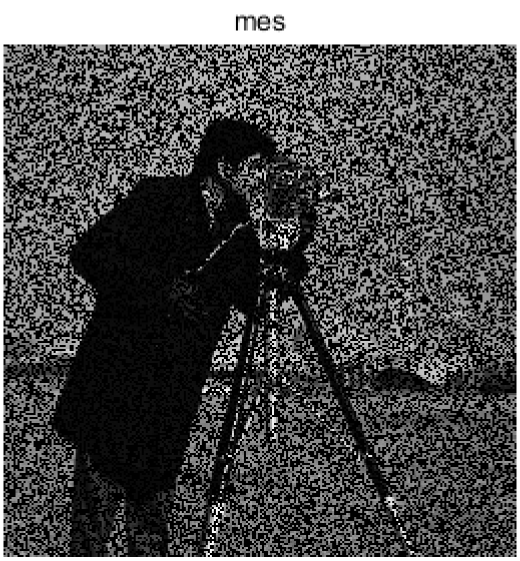
```
half_bandwidth = 0
```

利用编码算子**A**对图像进行二值编码

```
var_mes = optA*var_im;
length(var_mes)
```

```
ans = 65536
```

```
mes = reshape(var_mes, [height, width]);
imshow(mes, []);
title('mes');
```



```
psnr_mes = psnr(mes, im, 255)
```

```
psnr_mes = 8.5790
```

```
ssim_mes = ssim(mes, im)
```

```
ssim_mes = 0.0921
```

ADMM滤波

利用压缩感知加入稀疏约束

$x = \text{argmin}\ \|\text{Ax} - y\|_2^2 + \lambda\|x\|_{\text{TV}}$

利用ADMM, 对问题进行分解

$x, v, u = \text{argmin}\ \|\text{Ax} - y\|_2^2 + \lambda\|v\|_{\text{TV}} + u^T(x - v) + \frac{\rho}{2}\|x - v\|_2^2$

可以根据KKT条件, 得到最优化的目标方程

$$x^{(t+1)} = \underset{x}{\text{argmin}}\ \|\text{Ax} - y\|_2^2 + \frac{\rho}{2}\left\|x - \left(v^{(t)} + \frac{1}{\rho}u^{(t)}\right)\right\|_2^2$$

$$v^{(t+1)} = \underset{v}{\text{argmin}}\ \lambda\|v\|_{\text{TV}} + \frac{\rho}{2}\left\|v - \left(x^{(t+1)} - \frac{1}{\rho}u^{(t)}\right)\right\|_2^2$$

$$u^{(t+1)} = u^{(t)} - \rho(x^{(t+1)} - v^{(t+1)})$$

经过计算, 得到迭代公式

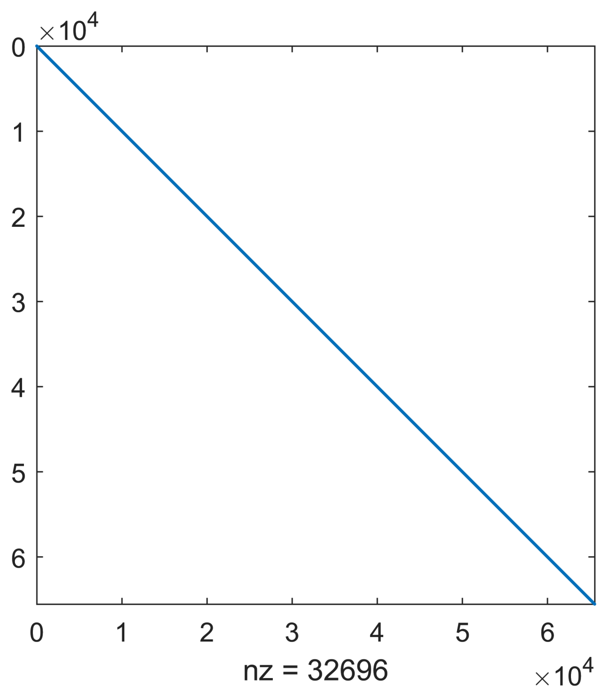
$$x^{(t+1)} = (A^TA + \rho)^{-1}(\rho v^{(t)} + u^{(t)} + A^Ty)$$

$$v^{(t+1)} = \text{Denoiser}_{\text{TV}}\left(x^{(t+1)} - \frac{1}{\rho}u^{(t)}\right)$$

$$u^{(t+1)} = u^{(t)} - \rho(x^{(t+1)} - v^{(t+1)})$$

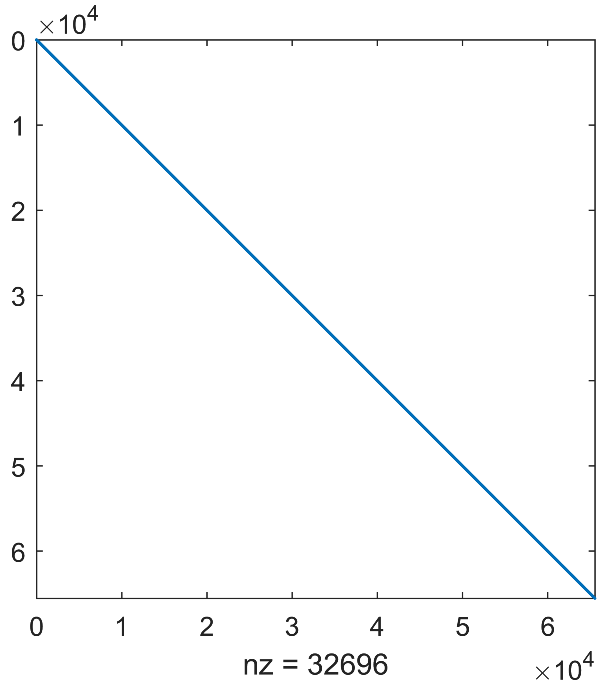
定义迭代过程的常量

```
A = optA;
At = optA';
spy(At);
```




```
AtA = At*A;
```

```
spy(AtA);
```




```
AtA = full(diag(AtA));
lambda = 10;
rho = 0.01;
y = var_mes;
iters_max = 64;
size_img = [height, width];
src = im;
```

定义迭代过程的变量初值

```
xt = 255*rand(size(src(:)));
vt = xt;
ut = 0;
```

运行迭代

```
fprintf('Begin ADMM algorithm restoration\n');
```

```
Begin ADMM algorithm restoration
```

```
for i=1:iters_max
    xt = (rho*vt+ut + At*y)./(AtA+rho);
    vt = TV_denoising(reshape(xt-1/rho*ut,size_img),lambda,5);
    psnr_ = psnr(vt, src, 255);
    ssim_ = ssim(vt, src);
    vt = vt(:);
    ut_ = xt-vt;
    ut = ut - rho*ut_;
    fprintf(['Iters=',num2str(i,'%02d'),'   ', ...
        'ut_=', num2str(sum(abs(ut_)),'%.4e'),'   ', ...
```

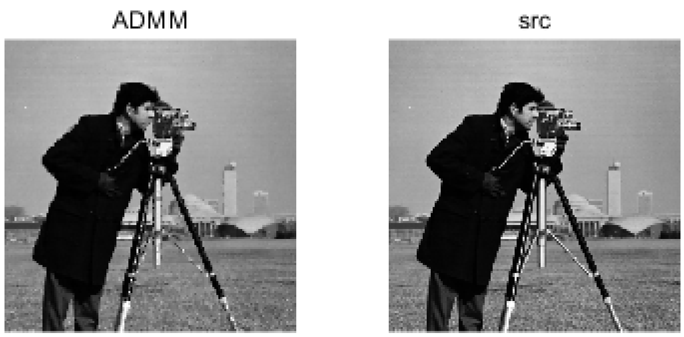
```
        'PSNR=',num2str(psnr_,'%2.4f'),'dB  ', ...
        'SSIM=',num2str(ssim_),'\n']);

end
```

```
Iters=01  ut_=3.3482e+05  PSNR=11.9435dB  SSIM=0.082143
Iters=02  ut_=4.1803e+04  PSNR=12.5228dB  SSIM=0.098387
Iters=03  ut_=3.3458e+04  PSNR=13.1019dB  SSIM=0.11128
Iters=04  ut_=3.1420e+04  PSNR=13.6891dB  SSIM=0.12548
Iters=05  ut_=3.0502e+04  PSNR=14.2827dB  SSIM=0.14143
Iters=06  ut_=3.0467e+04  PSNR=14.8802dB  SSIM=0.15929
Iters=07  ut_=3.0522e+04  PSNR=15.4790dB  SSIM=0.17909
Iters=08  ut_=3.0952e+04  PSNR=16.0765dB  SSIM=0.20081
Iters=09  ut_=3.0329e+04  PSNR=16.6713dB  SSIM=0.22414
Iters=10  ut_=2.8998e+04  PSNR=17.2617dB  SSIM=0.24857
Iters=11  ut_=2.7151e+04  PSNR=17.8457dB  SSIM=0.27324
Iters=12  ut_=2.5479e+04  PSNR=18.4212dB  SSIM=0.29822
Iters=13  ut_=2.3731e+04  PSNR=18.9861dB  SSIM=0.32427
Iters=14  ut_=2.2187e+04  PSNR=19.5378dB  SSIM=0.35193
Iters=15  ut_=2.0943e+04  PSNR=20.0745dB  SSIM=0.38151
Iters=16  ut_=1.9933e+04  PSNR=20.5952dB  SSIM=0.41275
Iters=17  ut_=1.8452e+04  PSNR=21.0995dB  SSIM=0.44489
Iters=18  ut_=1.7026e+04  PSNR=21.5866dB  SSIM=0.47609
Iters=19  ut_=1.5616e+04  PSNR=22.0555dB  SSIM=0.50551
Iters=20  ut_=1.4199e+04  PSNR=22.5048dB  SSIM=0.53336
Iters=21  ut_=1.3021e+04  PSNR=22.9334dB  SSIM=0.55857
Iters=22  ut_=1.1878e+04  PSNR=23.3408dB  SSIM=0.57984
Iters=23  ut_=1.0765e+04  PSNR=23.7259dB  SSIM=0.59688
Iters=24  ut_=9.6219e+03  PSNR=24.0883dB  SSIM=0.61081
Iters=25  ut_=8.3449e+03  PSNR=24.4276dB  SSIM=0.62242
Iters=26  ut_=7.2282e+03  PSNR=24.7425dB  SSIM=0.63271
Iters=27  ut_=6.5758e+03  PSNR=25.0311dB  SSIM=0.64223
Iters=28  ut_=5.9913e+03  PSNR=25.2926dB  SSIM=0.65141
Iters=29  ut_=5.5843e+03  PSNR=25.5266dB  SSIM=0.66043
Iters=30  ut_=5.2134e+03  PSNR=25.7336dB  SSIM=0.66948
Iters=31  ut_=4.8387e+03  PSNR=25.9155dB  SSIM=0.67816
Iters=32  ut_=4.4064e+03  PSNR=26.0745dB  SSIM=0.68669
Iters=33  ut_=4.0257e+03  PSNR=26.2127dB  SSIM=0.69476
Iters=34  ut_=3.5690e+03  PSNR=26.3321dB  SSIM=0.70273
Iters=35  ut_=3.2061e+03  PSNR=26.4346dB  SSIM=0.71077
Iters=36  ut_=2.9364e+03  PSNR=26.5220dB  SSIM=0.71808
Iters=37  ut_=2.6158e+03  PSNR=26.5967dB  SSIM=0.72505
Iters=38  ut_=2.3610e+03  PSNR=26.6602dB  SSIM=0.73131
Iters=39  ut_=2.1039e+03  PSNR=26.7142dB  SSIM=0.73656
Iters=40  ut_=1.8566e+03  PSNR=26.7601dB  SSIM=0.74121
Iters=41  ut_=1.6509e+03  PSNR=26.7992dB  SSIM=0.7452
Iters=42  ut_=1.4704e+03  PSNR=26.8328dB  SSIM=0.74851
Iters=43  ut_=1.2799e+03  PSNR=26.8618dB  SSIM=0.75139
Iters=44  ut_=1.1310e+03  PSNR=26.8869dB  SSIM=0.75379
Iters=45  ut_=1.0182e+03  PSNR=26.9089dB  SSIM=0.75564
Iters=46  ut_=8.9160e+02  PSNR=26.9284dB  SSIM=0.75703
Iters=47  ut_=7.7158e+02  PSNR=26.9457dB  SSIM=0.75809
Iters=48  ut_=6.5878e+02  PSNR=26.9611dB  SSIM=0.75904
Iters=49  ut_=5.7549e+02  PSNR=26.9746dB  SSIM=0.75996
Iters=50  ut_=5.2142e+02  PSNR=26.9862dB  SSIM=0.76082
Iters=51  ut_=4.7966e+02  PSNR=26.9962dB  SSIM=0.76159
Iters=52  ut_=4.3677e+02  PSNR=27.0049dB  SSIM=0.76219
Iters=53  ut_=3.9843e+02  PSNR=27.0125dB  SSIM=0.76264
Iters=54  ut_=3.6000e+02  PSNR=27.0190dB  SSIM=0.76298
Iters=55  ut_=3.2756e+02  PSNR=27.0246dB  SSIM=0.76327
Iters=56  ut_=2.9891e+02  PSNR=27.0293dB  SSIM=0.76352
Iters=57  ut_=2.6629e+02  PSNR=27.0334dB  SSIM=0.76374
Iters=58  ut_=2.4084e+02  PSNR=27.0370dB  SSIM=0.76391
Iters=59  ut_=2.1363e+02  PSNR=27.0400dB  SSIM=0.76403
```

```
Iters=60  ut_=1.9217e+02  PSNR=27.0427dB  SSIM=0.76411
Iters=61  ut_=1.7433e+02  PSNR=27.0450dB  SSIM=0.76418
Iters=62  ut_=1.5789e+02  PSNR=27.0472dB  SSIM=0.76423
Iters=63  ut_=1.4298e+02  PSNR=27.0491dB  SSIM=0.76429
Iters=64  ut_=1.2836e+02  PSNR=27.0509dB  SSIM=0.76434
```

对比结果

```
subplot(1,2,1);
imshow(reshape(xt,size_img),[]); title('ADMM');
subplot(1,2,2);
imshow(src,[]); title('src');
```




```
fprintf(['Iters=',num2str(i,'%02d'),'  ', ...
    'PSNR=',num2str(psnr_),'dB  ', ...
    'SSIM=',num2str(ssim_),'\n']);
```

```
Iters=64  PSNR=27.0509dB  SSIM=0.76434
```

GAP滤波

利用压缩感知加入稀疏约束

$x = \text{argmin} \|\text{Ax} - y\|_2^2 + \lambda\|x\|_{\text{TV}}$

利用ADMM, 对问题进行分解

$x, v = \text{argmin} \|\text{Ax} - y\|_2^2 + \lambda\|v\|_{\text{TV}}$

可以根据KKT条件, 得到最优化的迭代公式

$$x^{(t+1)} = v^{(t)} + \Delta(A^TA)^{-1}A^T(y - Av^{(t)})$$
$$v^{(t+1)} = \text{Denoiser}_{TV}(x^{(t+1)})$$

定义迭代过程的常量

```
A = optA;
At = optA';
spy(At);
AtA = At*A;
spy(AtA);
AtA = full(diag(AtA));
lambda = 10;
delta = 1;
y = var_mes;
iters_max = 64;
size_img = [height, width];
src = im;
```

定义迭代过程的变量初值

```
xt = 255*rand(size(src(:)));
vt = xt;
```

运行迭代

```
fprintf('Begin GAP algorithm restoration\n');
```

```
Begin GAP algorithm restoration
```

```
for i=1:iters_max
    xt = vt + delta*At*(y-A*vt)./(AtA+eps);
    vt = TV_denoising(reshape(xt,size_img),lambda,5);
    psnr_ = psnr(vt, src, 255);
    ssim_ = ssim(vt, src);
    vt = vt(:);
    fprintf(['Iters=',num2str(i,'%02d'),'   ', ...
        'PSNR=',num2str(psnr_,'%2.4f'),'dB   ', ...
        'SSIM=',num2str(ssim_),'\n']);
end
```

```
Iters=01   PSNR=12.0133dB   SSIM=0.082309
Iters=02   PSNR=12.5878dB   SSIM=0.092673
Iters=03   PSNR=13.1717dB   SSIM=0.10429
Iters=04   PSNR=13.7636dB   SSIM=0.11736
Iters=05   PSNR=14.3614dB   SSIM=0.13204
Iters=06   PSNR=14.9626dB   SSIM=0.1484
Iters=07   PSNR=15.5646dB   SSIM=0.16643
Iters=08   PSNR=16.1643dB   SSIM=0.18609
Iters=09   PSNR=16.7597dB   SSIM=0.20707
Iters=10   PSNR=17.3480dB   SSIM=0.22857
Iters=11   PSNR=17.9270dB   SSIM=0.24989
Iters=12   PSNR=18.4946dB   SSIM=0.27095
Iters=13   PSNR=19.0496dB   SSIM=0.29198
Iters=14   PSNR=19.5902dB   SSIM=0.31285
Iters=15   PSNR=20.1153dB   SSIM=0.33319
```

```
Iters=16  PSNR=20.6242dB  SSIM=0.35335
Iters=17  PSNR=21.1165dB  SSIM=0.37347
Iters=18  PSNR=21.5919dB  SSIM=0.3933
Iters=19  PSNR=22.0488dB  SSIM=0.41242
Iters=20  PSNR=22.4858dB  SSIM=0.43024
Iters=21  PSNR=22.9018dB  SSIM=0.44666
Iters=22  PSNR=23.2958dB  SSIM=0.46121
Iters=23  PSNR=23.6666dB  SSIM=0.47397
Iters=24  PSNR=24.0135dB  SSIM=0.48513
Iters=25  PSNR=24.3359dB  SSIM=0.49493
Iters=26  PSNR=24.6324dB  SSIM=0.50367
Iters=27  PSNR=24.9027dB  SSIM=0.51154
Iters=28  PSNR=25.1468dB  SSIM=0.51869
Iters=29  PSNR=25.3658dB  SSIM=0.52537
Iters=30  PSNR=25.5607dB  SSIM=0.53166
Iters=31  PSNR=25.7330dB  SSIM=0.53769
Iters=32  PSNR=25.8840dB  SSIM=0.54364
Iters=33  PSNR=26.0152dB  SSIM=0.54959
Iters=34  PSNR=26.1282dB  SSIM=0.5555
Iters=35  PSNR=26.2248dB  SSIM=0.56133
Iters=36  PSNR=26.3074dB  SSIM=0.56702
Iters=37  PSNR=26.3776dB  SSIM=0.57247
Iters=38  PSNR=26.4374dB  SSIM=0.57751
Iters=39  PSNR=26.4880dB  SSIM=0.58213
Iters=40  PSNR=26.5308dB  SSIM=0.58628
Iters=41  PSNR=26.5670dB  SSIM=0.5899
Iters=42  PSNR=26.5979dB  SSIM=0.59295
Iters=43  PSNR=26.6244dB  SSIM=0.59541
Iters=44  PSNR=26.6472dB  SSIM=0.59737
Iters=45  PSNR=26.6669dB  SSIM=0.59898
Iters=46  PSNR=26.6839dB  SSIM=0.60037
Iters=47  PSNR=26.6987dB  SSIM=0.60153
Iters=48  PSNR=26.7116dB  SSIM=0.60248
Iters=49  PSNR=26.7228dB  SSIM=0.60326
Iters=50  PSNR=26.7326dB  SSIM=0.6039
Iters=51  PSNR=26.7411dB  SSIM=0.60446
Iters=52  PSNR=26.7486dB  SSIM=0.6049
Iters=53  PSNR=26.7554dB  SSIM=0.60523
Iters=54  PSNR=26.7615dB  SSIM=0.60548
Iters=55  PSNR=26.7670dB  SSIM=0.60567
Iters=56  PSNR=26.7720dB  SSIM=0.60581
Iters=57  PSNR=26.7765dB  SSIM=0.60593
Iters=58  PSNR=26.7806dB  SSIM=0.60603
Iters=59  PSNR=26.7842dB  SSIM=0.60611
Iters=60  PSNR=26.7876dB  SSIM=0.60619
Iters=61  PSNR=26.7906dB  SSIM=0.60625
Iters=62  PSNR=26.7933dB  SSIM=0.60631
Iters=63  PSNR=26.7957dB  SSIM=0.60636
Iters=64  PSNR=26.7978dB  SSIM=0.60641
```

对比结果

```
subplot(1,2,1);
imshow(reshape(xt,size_img),[]); title('GAP');
subplot(1,2,2);
imshow(src,[]); title('src');
```


```
fprintf(['Iters=',num2str(i,'%02d'),'   ', ...
    'PSNR=',num2str(psnr_),'dB   ', ...
    'SSIM=',num2str(ssim_),'\n']);
```

```
Iters=64   PSNR=26.7978dB   SSIM=0.60641
```