CSE 559A/Fall 2020. Problem Set 2 Solution Key.

It is a violation of the academic integrity policy to scan, copy, or otherwise share these solutions

1 (a)

$$x = \text{sign}(y) \max\left(0., |y| - \frac{\lambda}{2}\right)$$

The key observation in this question is that the derivative of $|x|$ wrt x is discontinuous at 0. The derivative is 1 when $x > 0$, and -1 when $x < 0$. When $y > \lambda/2$ or $y < \lambda/2$, you can find a solution for the derivative being 0, and that is indeed the minima. But when y lies between $(-\lambda/2, \lambda/2)$, there is no value of x for which the derivative is 0. However, the derivative does change sign at 0, and that's what defines a minima or a maxima (the function was going down, and then it started going up). And you can see then that for these values of y, $x = 0$ yields the smallest value of the cost. Therefore, denoising with an absolute value regularizer yields "sparse" solutions—instead of just scaling down the observed coefficients, it will set them exactly to 0 if they're small enough.

(b)

```
def denoise_coeff(y,lmbda):
    return np.sign(y)*np.maximum(0,np.abs(y)-lmbda/2)
```

2 (a)

```
def balance2a(img):
    c = np.mean(np.reshape(img,[-1,3]),axis=0)
    c = 1/c
    c = c / np.sum(c)*3
    return img*c
```

(b)

```
def balance2b(img):
    c = np.zeros((3))
    for i in range(3):
        v = img[:,:,i].flatten()
        v = np.sort(v)
        v = v[round(0.9*v.shape[0]):]
        c[i] = 1/np.mean(v)
    c = c / np.sum(c)*3
    return img*c
```

The images produced by (b) are less affected by dominant colors. For example, for the third image, *gray world* thinks the illumination has a green hue because there are so many green pixels in the image, and this biases the mean, while the method in (b) concentrates on the brightest pixels (which are more likely to be achromatic). As a result, the leaves in (b) appear closer to their true color of green.

3 (a)

```
def pstereo_n(imgs, L, mask):
    imv = []
    yx = np.where(mask>0)
    for i in range(len(imgs)):
        imv = imv + [np.mean(imgs[i],axis=2)[yx[0],yx[1]]]
    imv = np.stack(imv,axis=-1)
    imv = np.matmul(imv,L)
    nrm = np.linalg.solve(np.matmul(L.T,L),imv.T).T
    nrm = nrm / np.sqrt(np.maximum(1e-8,np.sum(nrm**2,axis=1,keepdims=True)))
    imn = np.zeros(imgs[0].shape)
    imn[yx[0],yx[1],:] = nrm
```

```
    return imn
```

(b)

```
def pstereo_alb(imgs, nrm, L, mask):
    yx = np.where(mask > 0)
    num = np.zeros((len(yx[0]),3))
    den = np.zeros((len(yx[0]),3))
    nrm = nrm[yx[0],yx[1],:]
    for i in range(len(imgs)):
        ai = np.maximum(0.,np.sum(nrm*L[i,:],axis=1,keepdims=True))
        den = den + ai*ai
        num = num + imgs[i][yx[0],yx[1],:]*ai
    alb = num/np.maximum(1e-8,den)
    imalb = np.zeros(imgs[0].shape)
    imalb[yx[0],yx[1],:] = alb

    return imalb
```

4.

```
def ntod(nrm, mask, lmda):
    denom = nrm[:,:,2]*mask + (1-mask)
    dx = -nrm[:,:,0]*mask / denom
    dy = -nrm[:,:,1]*mask / denom

    gxf = np.zeros(dx.shape); gyf = np.zeros(dx.shape); lpf = np.zeros(dx.shape)
    gxf[0,1] = -0.5; gxf[0,-1] = 0.5; gyf[1,0] = 0.5; gyf[-1,0] = -0.5
    lpf[0:2,0:2] = -1/9; lpf[-1,0:2] = -1/9; lpf[0:2,-1] = -1/9; lpf[-1,-1] = -1/9; lpf[0,0] = 8/9

    gxf = np.fft.fft2(gxf); gyf = np.fft.fft2(gyf); lpf = np.fft.fft2(lpf)
    dx = np.fft.fft2(dx); dy = np.fft.fft2(dy)

    znf = np.conj(gxf)*dx + np.conj(gyf)*dy
    zdf = np.absolute(gxf)**2 + np.absolute(gyf)**2 + lmda * np.absolute(lpf)**2
    zf = znf / np.maximum(1e-12,zdf); zf[0,0] = 0
    Z = np.real(np.fft.ifft2(zf))

    return Z
```

5.

```
fx = np.float32([[0.5,0,-0.5]]); fy = np.float32([[-0.5],[0],[0.5]])
fr = -np.ones((3,3),dtype=np.float32)/9.; fr[1,1] = 8./9.

def QxP(p,wts,lmda):
    return conv2(conv2(p,fx,'same')*wts,-fx,'same') + conv2(conv2(p,fy,'same')*wts,-fy,'same') + \
         lmda * conv2(conv2(p,fr,'same'),fr,'same')

def ntod(nrm, mask, lmda):
    denom = nrm[:,:,2]*mask + (1-mask)
    dx = -nrm[:,:,0]*mask / denom; dy = -nrm[:,:,1]*mask / denom
    wts = mask*(nrm[:,:,2]**2)

    b = conv2(dx*wts,-fx,'same') + conv2(dy*wts,-fy,'same')
    r = b; p = r.copy(); Z = np.zeros(dx.shape,dtype=np.float32);
    r2 = np.sum(r[:]**2)
```

```
for it in range(100):
    Qp = QxP(p,wts,lmda)
    alpha = r2 / np.sum(p[:]*Qp[:])
    Z = Z + alpha*p
    r = r - alpha*Qp
    r2new = np.sum(r[:]**2)
    beta = r2new / r2
    p = r + beta*p
    r2 = r2new
return Z
```