4.1

a) $C = \begin{bmatrix} x_{11} & 0 & \cdots & 0 \\ 0 & x_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & x_{NN} \end{bmatrix}$, $Z = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1N} \\ z_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ z_{N1} & \cdots & \cdots & z_{NN} \end{bmatrix}$.

$$Z^T C Z = \begin{bmatrix} z_{11} \cdot x_{11} & \cdots & \cdots & 0 \\ 0 & z_{22} \cdot x_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & z_{NN} \cdot x_{NN} \end{bmatrix} Z$$

$$= \begin{bmatrix} z_{11}^2 x_{11} & \cdots & \cdots & 0 \\ \vdots & z_{22}^2 x_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & z_{NN}^2 x_{NN} \end{bmatrix}$$

∵ C is non-negative, thus $\forall N, x_{11} \cdots x_{NN} > 0$,

Thus, $z_{11}^2 x_{11} \cdots z_{NN}^2 x_{NN} > 0$ for all N.

∴ if C non negative, $Z^T C Z \geq 0, \forall Z$.

b) Conversely, as above

if $\forall N, z_{11}^2 x_{11} \cdots z_{NN}^2 x_{NN} > 0$, $x_{11} \cdots x_{NN} > 0$.

∴ its also true.

c). For $C = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, eigenvalues are $\lambda_1 = 0$, $\lambda_2 = 2$.

∵ C is non negative, thus g convex holds.

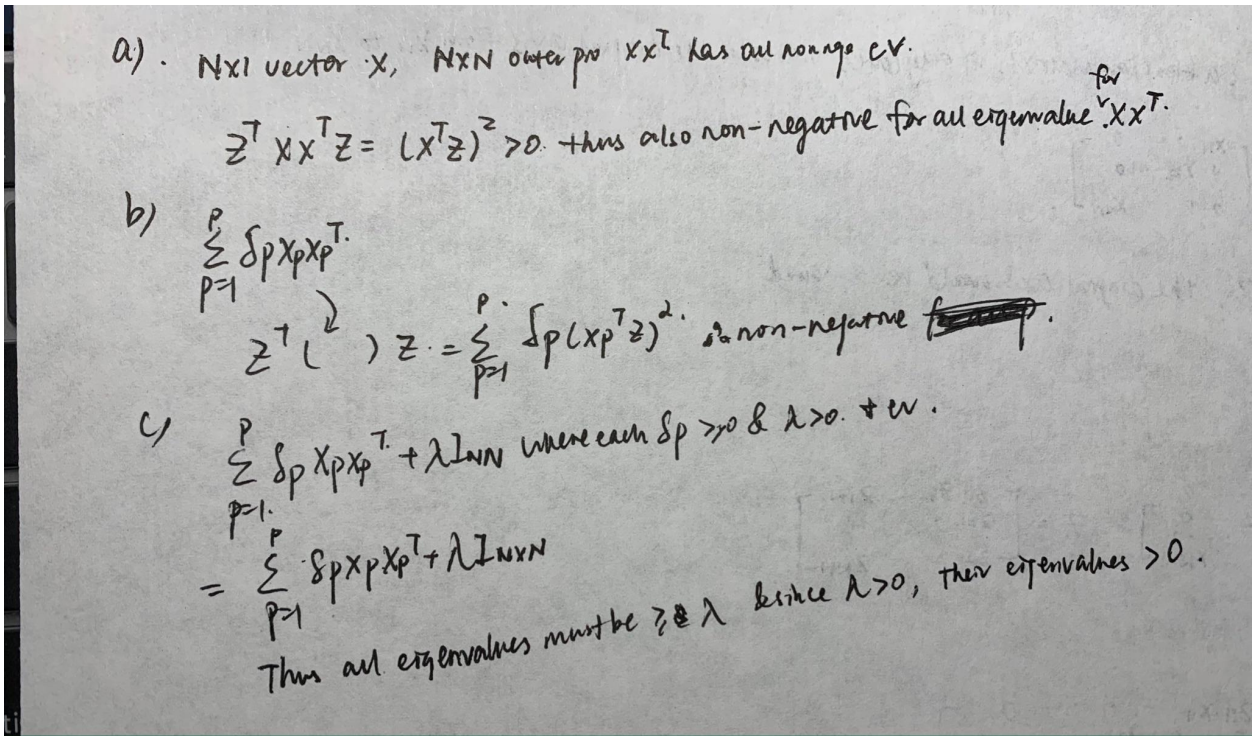
d). Say the eigenvalues here are $e_1 \cdots e_N$.

$\forall N$, we have $c_{11}^2 \cdot e_1 + \lambda \cdots c_{NN}^2 (e_N + \lambda)$.

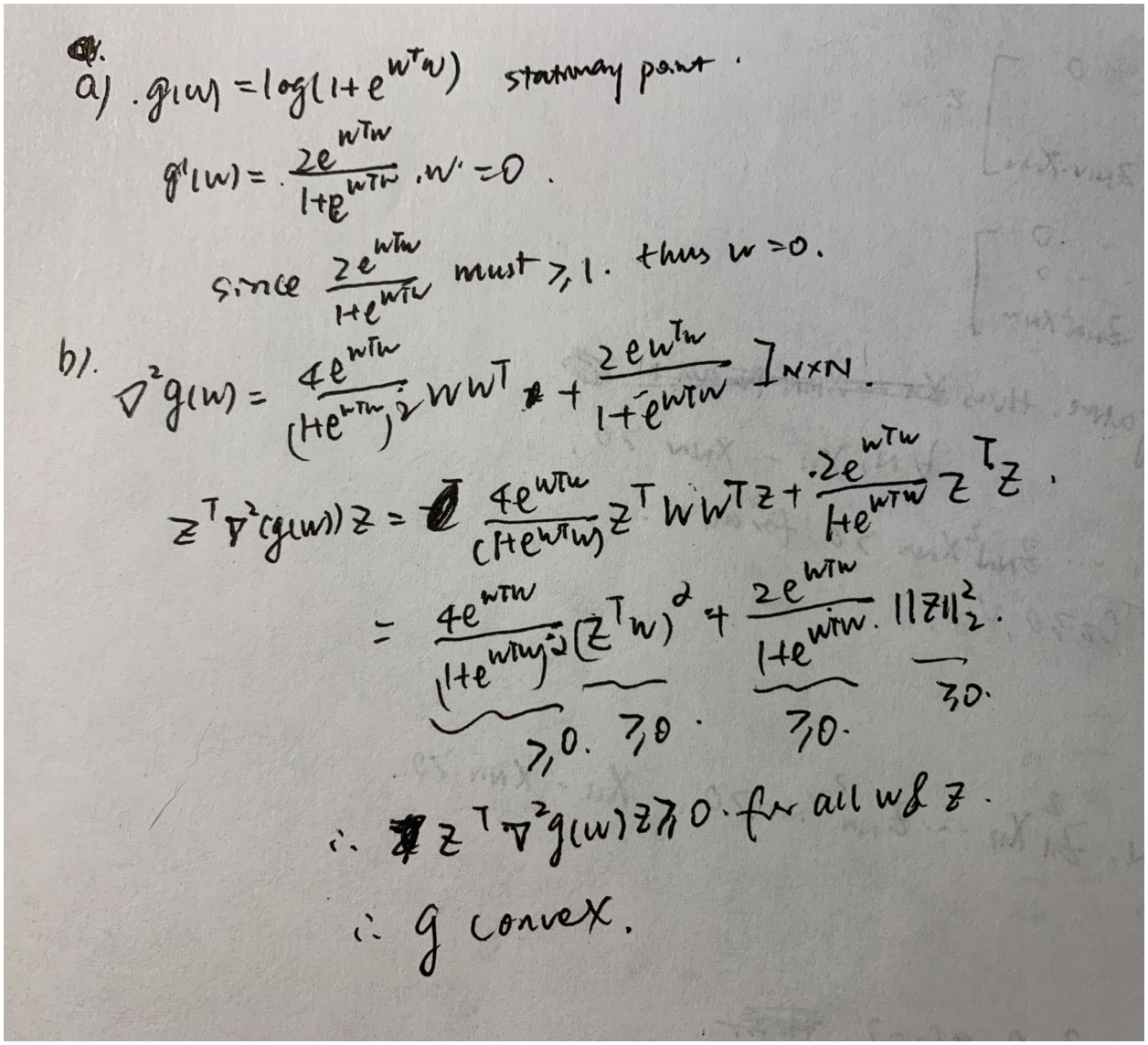
if $\forall N, e_N \geq -\lambda$, it would be positive.

Thus, λ should be the absolute value of the largest eigenvalue of C.

4.2


a). Nx1 vector X, NxN outer pro XX^T has all nonnge eV.
$z^T XX^T z = (X^T z)^2 \geq 0$. thus also non-negative for all eigenvalue for XX^T.
b) $\sum_{p=1}^{P} \delta_p X_p X_p^T$.
$z^T (\quad) z = \sum_{p=1}^{P} \delta_p (X_p^T z)^2$. is non-negative
c) $\sum_{p=1}^{P} \delta_p X_p X_p^T + \lambda I_{NN}$ where each $\delta_p \geq 0$ & $\lambda > 0$. + eV.
$= \sum_{p=1}^{P} \delta_p X_p X_p^T + \lambda I_{NxN}$
Thus all eigenvalues must be $\geq \lambda$ because $\lambda > 0$, then eigenvalues > 0.

4.5

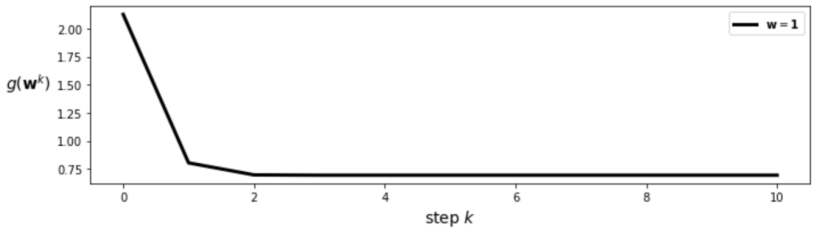

a) $g(w) = \log(1+e^{w^T w})$ stationary point.
$g'(w) = \frac{2e^{w^T w}}{1+e^{w^T w}} \cdot w = 0$.
since $\frac{2e^{w^T w}}{1+e^{w^T w}}$ must ≥ 1. thus $w = 0$.
b) $\nabla^2 g(w) = \frac{4e^{w^T w}}{(1+e^{w^T w})^2} w w^T + \frac{2e^{w^T w}}{1+e^{w^T w}} I_{N\times N}$.
$z^T \nabla^2 (g(w)) z = \frac{4e^{w^T w}}{(1+e^{w^T w})^2} z^T w w^T z + \frac{2e^{w^T w}}{1+e^{w^T w}} z^T z$.
$= \frac{4e^{w^T w}}{(1+e^{w^T w})^2} (z^T w)^2 + \frac{2e^{w^T w}}{1+e^{w^T w}} \cdot \|z\|_2^2$.
≥ 0. ≥ 0. ≥ 0. ≥ 0.
$\therefore z^T \nabla^2 g(w) z \geq 0$. for all w & z.
$\therefore g$ convex.

C.

```
def newtons_method(g,maxx,w,**args):
    gradient = grad(g)
    h = hessian(g)
    e = 10**(-10)
    if 'epsilon' in args:
        e = args['epsilon']
    wh = [w]
    ch = [g(w)]
    for k in range(maxx):
        grade = gradient(w)
        he = h(w)
        he.shape = (int((np.size(he))**(0.5)),int((np.size(he))**(0.5)))
        a = he + e*np.eye(w.size)
        b = grade
        w = np.linalg.solve(a,np.dot(a,w) - b)
        wh.append(w)
        ch.append(g(w))
    return wh,ch
```

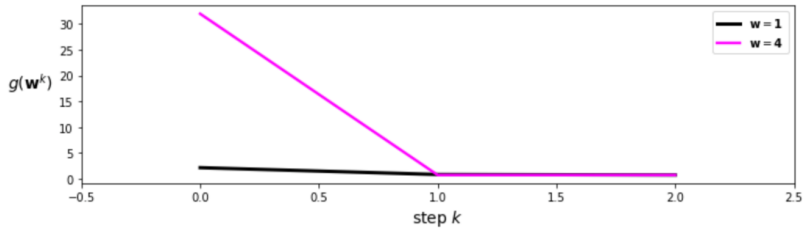
In [41]:
```
g = lambda w: np.log(1 + np.exp(np.dot(w.T,w)))
w = np.ones((2,)); maxx = 10;
wh,ch = newtons_method(g,maxx,w)
static_plotter.plot_cost_histories([ch],start = 0,points = False,labels= [r'$\mathbf{w}=\mathbf{1}$'])
```


In [41]:
```
g = lambda w: np.log(1 + np.exp(np.dot(w.T,w)))
w = np.ones((2,)); maxx = 10;
wh,ch = newtons_method(g,maxx,w)
static_plotter.plot_cost_histories([ch],start = 0,points = False,labels= [r'$\mathbf{w}=\mathbf{1}$'])
```



D

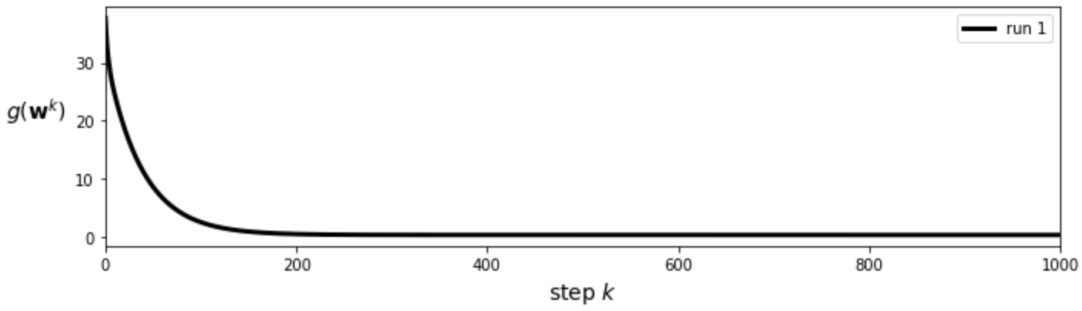
In [42]:
```
w = np.ones((2,)); maxx = 2;
wh,ch = newtons_method(g,maxx,w)
w = 4*np.ones((2,)); maxx = 2;
wh2,ch2 = newtons_method(g,maxx,w)
static_plotter.plot_cost_histories([ch,ch2],start = 0,points= False,
                                   labels = [r'$\mathbf{w}=\mathbf{1}$',r'$\mathbf{w}=\mathbf{4}$'])
```



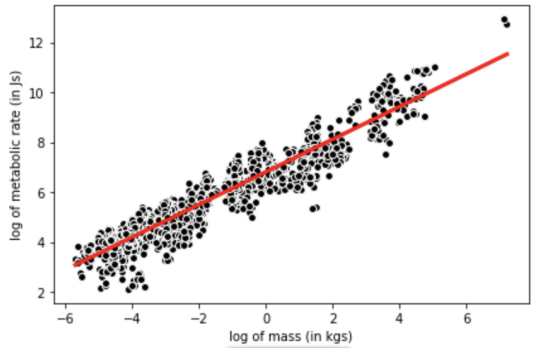
5.2

In [71]:
```
csvname = 'mlrefined_exercises/ed_2/mlrefined_datasets/superlearn_datasets/kleibers_law_data.csv'
data = np.loadtxt(csvname,delimiter=',')
x = data[:-1,:]
y = data[-1:,:]
x = np.log(x)
y = np.log(y)

def new_gradient_descent(g,a,maxx,w):
    gradient = grad(g)
    wh = [w]
    ch = [g(w)]
    for k in range(maxx):
        grade = gradient(w)
        w = w - a*grade
        wh.append(w)
        ch.append(g(w))
    return wh,ch
def model(x,w):
    a = w[0] + np.dot(x.T,w[1:])
    return a.T
def least(w):
    cost = np.sum((model(x,w) - y)**2)
    return cost/float(np.size(y))
g = least
w = 0.1*np.random.randn(2,1)
maxx = 1000
a = 10**(-2)
wh, ch = new_gradient_descent(g,a,maxx,w)
static_plotter.plot_cost_histories([ch],start = 0,points = False,labels= ['run 1'])
```



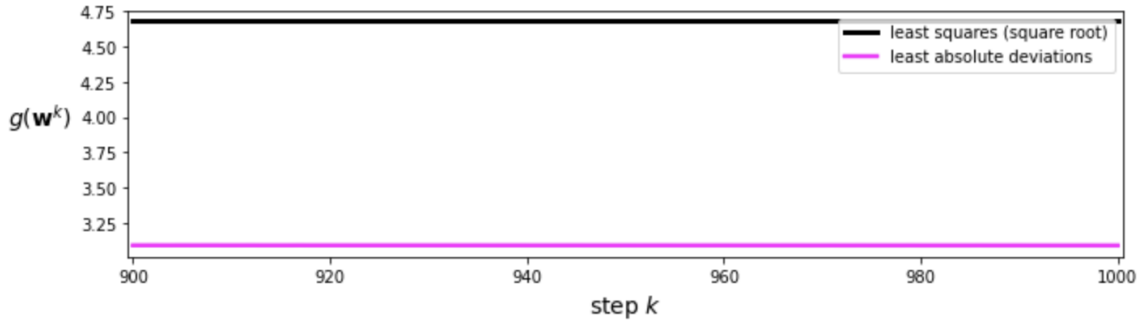
In [73]:
```
s = np.linspace(np.min(x),np.max(x))
w = wh[-1]
t = w[0] + w[1]*s
figure = plt.figure()
plt.plot(s,t,linewidth = 3,color = 'r')
plt.scatter(x,y,linewidth = 1,c='k',edgecolor='w')
plt.xlabel('log of mass (in kgs)')
plt.ylabel('log of metabolic rate (in Js)')
plt.show()
```



5.9

In [74]:
```
def normalize(x):
    xmeans = np.nanmean(x,axis = 1)[:,np.newaxis]
    xstds = np.nanstd(x,axis = 1)[:,np.newaxis]
    ind = np.argwhere(xstds < 10**(-2))
    if len(ind) > 0:
        ind = [v[0] for v in ind]
        ad = np.zeros((xstds.shape))
        ad[ind] = 1.0
        xstds += ad
    ind = np.argwhere(np.isnan(x) == True)
    for i in ind:
        x[i[0],i[1]] = xmeans[i[0]]
    return lambda data: (data - xmeans)/xstds,lambda data: data*xstds + xmeans
```

In [82]:
```
csvname = 'mlrefined_exercises/ed_2/mlrefined_datasets/superlearn_datasets/boston_housing.csv'
data = np.loadtxt(csvname,delimiter=',')
x = data[:-1,:]
y = data[-1:,:]
norm,inverse_norm = normalize(x)
x = norm(x)
def model(x,w):
    a = w[0] + np.dot(x.T,w[1:])
    return a.T
def ls(w):
    cost = np.sum((model(x,w) - y)**2)
    return cost/float(np.size(y))
def lad(w):
    cost = np.sum(np.abs(model(x,w) - y))
    return cost/float(np.size(y))
g = ls
w = 0.1*np.random.randn(x.shape[0]+1,1)
maxx = 1000
a = 10**(-1)
wh1,ch1 = optimizers.gradient_descent(g,alpha_choice,max_its,w)
ch1 = [c**(0.5) for c in ch1]
g = lad
wh2,ch2 = optimizers.gradient_descent(g,a,maxx,w)
static_plotter.plot_cost_histories([ch1,ch2],start = 900,points = False,
                                   labels = ['least squares (square root)','least absolute deviations'])
```



In [83]:
```
csvname = 'mlrefined_exercises/ed_2/mlrefined_datasets/superlearn_datasets/auto_data.csv'
data = np.loadtxt(csvname,delimiter=',')
x = data[:-1,:]
y = data[-1:,:]
norm,inverse_norm = normalize(x)
x = norm(x)
g = ls;
w = 0.1*np.random.randn(x.shape[0]+1,1);
maxx = 1000;
a = 10**(-1);
wh1,ch1 = optimizers.gradient_descent(g,a,maxx,w)
ch1 = [c**(0.5) for c in ch1]
g = lad;
wh2,ch2 = optimizers.gradient_descent(g,alpha_choice,max_its,w)
static_plotter.plot_cost_histories([ch1,ch2],start = 900,points = False,
                                   labels = ['least squares (square root)','least absolute deviations'])
```