SOLUTION FOR ASSIGNMENT 3

Problem 3

Let $\mathbf{A}$ be an $n \times n$ real symmetric matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. The Rayleigh quotient defines a function $f : \mathcal{R}^n \to \mathcal{R}$ given by

$$f(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}.$$

It can be shown that the critical points of f are eigenvectors of $\mathbf{A}$, in particular

$$\lambda_1 = \min_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}$$

$$\lambda_n = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}$$

with the minimum and maximum occurring at the corresponding eigenvectors, $\mathbf{x}_1$ and $\mathbf{x}_n$.

(a) We can therefore compute the extreme eigenvectors and their corresponding eigenvalues using methods for optimization. To find $\mathbf{x}_1$, we can use unconstrained minimization method such as the steepest descent method.

In order to use the steepest descent method, we need to compute the gradient of f. Using Einstein convention we can write

$$f(\mathbf{x}) = \frac{x_i A_{i,j} x_j}{x_k x_k},$$

and so for $m = 1, 2, \cdots n$ we have

$$\frac{\partial f}{\partial x_m} = \frac{A_{m,j}x_j + x_i A_{i,m}}{x_k x_k} - \frac{x_i A_{i,j} x_j 2 x_m}{(x_k x_k)^2}.$$

Since $\mathbf{A}$ is a symmetric matrix, $A_{i,m} = A_{m,i}$, we find that

$$\nabla f(\mathbf{x}) = 2\left[\frac{\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} - \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{(\mathbf{x}^T\mathbf{x})^2}\mathbf{x}\right].$$

If we set the gradient to zero, we see that the optimal solutions are the eigenvectors of $\mathbf{A}$, and the optimal function values are the corresponding eigenvectors.

Notice also that the Rayleigh quotient does not change if $\mathbf{x}$ is replaced by $a\mathbf{x}$ where a is any nonzero scalar. Thus the eigenvectors $\mathbf{x}$ can only be determined up to a normalization constant.

This method is implemented in Matlab program CP6p10a.m. The function and its gradient are computed in Matlab functions fcnCP6p10a and gradCP6p10a, respectively. The starting point for steepest descent is chosen randomly. We use a variable `sign` to control the overall sign of matrix $\mathbf{A}$. To compute $\mathbf{x}_1$ we set `sign=1` since we want to perform a minimization. But to compute $\mathbf{x}_n$ we set `sign=-1` since we want to perform a maximization.

A typical run to compute $\mathbf{x}_1$ produces the following results:

```
>>   x1         x2         x3         f
  0.660228   0.341971   0.289726   7.120161
  0.043298  -0.188262   2.321432   0.884749
 -0.311244  -0.517178   2.301370   0.620216
 -0.083468  -0.767706   2.275875   0.583716
 -0.128353  -0.807231   2.260896   0.579487
 -0.101451  -0.834664   2.252629   0.578997
 -0.106635  -0.839175   2.250724   0.578941
 -0.103525  -0.842317   2.249700   0.578934
 -0.104124  -0.842837   2.249477   0.578933
it =
     8
eVector =
       -0.0433046166996218
         -0.350532743219326
          0.935548772701687
eValue =
          0.578933484185869
evec =
     -0.0431682042947913 -0.497425032351127 -0.866432249704755
       -0.350731446032482  0.819589100011505 -0.453057567982585
        0.935480603167125  0.284327354176152 -0.209842790596346
eval =
        0.578933385691052                  0                  0
                        0   2.13307447534853                  0
                        0                  0   7.28799213896042
>>
```

The computed results for the eigenvectors and the eigenvalues using Matlab's built-in `eig` function are given by `evec` and `eval`, respectively. We see that the agreements with our results computed using optimization are rather good.

A typical run to compute $\mathbf{x}_n$ gives the following results:

```
>>   x1          x2          x3          f
  0.172956   0.979747   0.271447  -4.171586
  3.280899   0.332458   0.627469  -6.585472
  3.148243   1.598308   0.650401  -7.281934
  3.118135   1.614738   0.755763  -7.287908
  3.112651   1.626945   0.752308  -7.287991
  3.112194   1.627150   0.753756  -7.287992
  3.112118   1.627318   0.753708  -7.287992
it =
     6
eVector =
         0.866434327969241
         0.453056118161441
         0.209837339653779
eValue =
          7.28799213874011
evec =
   -0.0431682042947913  -0.497425032351127  -0.866432249704755
    -0.350731446032482   0.819589100011505  -0.453057567982585
     0.935480603167125   0.284327354176152  -0.209842790596346
eval =
     0.578933385691052                   0                   0
                     0    2.13307447534853                   0
                     0                   0    7.28799213896042
>>
```

The results agree very well with those obtained using Matlab's `eig` function.

(b) Any normalization condition can be handled using constraints in the optimization. Here we will be using the Euclidean norm and so $\mathbf{x}^T\mathbf{x} = 1$. We have only one equality constraint and so function $g(\mathbf{x})$ is a scalar function, and so is the unknown Lagrange multiplier λ (not to be confused with the eigenvalue).

It turns out that a good way to impose this constraint is to let

$$g(\mathbf{x}) = \frac{1}{2}(\mathbf{x}^T\mathbf{x} - 1)^2.$$

The gradient with respect to $\mathbf{x}$ is

$$\nabla g(\mathbf{x}) = 2(\mathbf{x}^T\mathbf{x} - 1)\mathbf{x}.$$

Instead of working with f, we replace that by the Lagrange function

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}),$$

which depends on a total of four parameters. Its gradient in a space of dimension $n + 1 = 4$ is given by

$$\nabla \mathcal{L}(\mathbf{x}, \lambda) = \left[\begin{array}{c} 2\left[\dfrac{\mathbf{Ax}}{\mathbf{x}^T\mathbf{x}} - \dfrac{\mathbf{x}^T\mathbf{Ax}}{(\mathbf{x}^T\mathbf{x})^2}\mathbf{x} + \lambda(\mathbf{x}^T\mathbf{x} - 1)\mathbf{x}\right] \\ \frac{1}{2}(\mathbf{x}^T\mathbf{x} - 1)^2 \end{array} \right].$$

We set this gradient to zero to find the critical points. The vanishing of the fourth component of the gradient vector implies that $\mathbf{x}^T\mathbf{x} = 1$. Using this result in the remaining components, we see that $\mathbf{x}$ is an eigenvector of $\mathbf{A}$ and the corresponding eigenvalue is given by the Rayleigh quotient.

A typical run to compute $\mathbf{x}_1$ produces the following results:

```
>>    x1          x2          x3   LagrangeMultiplier f
  0.015009   0.767950   0.970845   0.990083   2.932855
 -0.485905   0.420468   1.004569   0.968882   1.497954
 -0.002664  -0.264726   0.928307   0.922970   0.622869
 -0.063362  -0.313426   0.923530   0.922800   0.582183
 -0.041245  -0.339017   0.936836   0.922559   0.579431
 -0.046911  -0.344692   0.935420   0.922558   0.579015
 -0.042935  -0.348797   0.936017   0.922556   0.578947
 -0.043847  -0.349729   0.935686   0.922556   0.578936
 -0.043130  -0.350398   0.935595   0.922556   0.578934
 -0.043289  -0.350559   0.935531   0.922556   0.578933
 -0.043162  -0.350673   0.935502   0.922556   0.578933
 -0.043190  -0.350701   0.935490   0.922556   0.578933
it =
    11
eVector =
       -0.0431895770878849
        -0.350701322179119
         0.935490360121179
LagrangeMultiplier =
         0.922555825727686
functionValue =
         0.578933388000384
eValue =
         0.578932792177206
evec =
     -0.0431682042947913  -0.497425032351127  -0.866432249704755
      -0.350731446032482   0.819589100011505  -0.453057567982585
       0.935480603167125   0.284327354176152  -0.209842790596346
eval =
       0.578933385691052                   0                   0
                       0    2.13307447534853                   0
                       0                   0    7.28799213896042
>>
```

A typical run to compute $\mathbf{x}_n$ gives the following results:

```
>>   x1         x2         x3   LagrangeMultiplier f
  0.933380   0.683332   0.212560   0.839238  -7.103503
  0.947555   0.484426   0.251870   0.829800  -7.268574
  0.905988   0.479191   0.205891   0.827159  -7.283123
  0.889576   0.462989   0.221467   0.826661  -7.286493
  0.879566   0.461346   0.208995   0.826465  -7.287498
  0.874313   0.456482   0.213748   0.826413  -7.287822
  0.871062   0.455954   0.209596   0.826392  -7.287932
  0.869237   0.454280   0.211226   0.826385  -7.287971
  0.868098   0.454097   0.209761   0.826383  -7.287984
  0.867445   0.453499   0.210341   0.826382  -7.287989
  0.867036   0.453434   0.209814   0.826381  -7.287991
  0.866800   0.453218   0.210024   0.826381  -7.287992
  0.866652   0.453194   0.209832   0.826381  -7.287992
  0.866566   0.453116   0.209909   0.826381  -7.287992
  0.866512   0.453107   0.209839   0.826381  -7.287992
it =
    14
eVector =
         0.866512072422619
         0.453107318641183
         0.209839029882331
LagrangeMultiplier =
         0.826381178569984
functionValue =
          7.28799212130127
eValue =
          7.28931732749741
evec =
     -0.0431682042947913 -0.497425032351127 -0.866432249704755
      -0.350731446032482  0.819589100011505 -0.453057567982585
       0.935480603167125  0.284327354176152 -0.209842790596346
eval =
       0.578933385691052                  0                  0
                       0   2.13307447534853                  0
                       0                  0   7.28799213896042
>>
```

For both $\mathbf{x}_1$ and $\mathbf{x}_n$, the results agree very well with those obtained directly using Matlab's `eig` function.

Also notice that there is no special significance to the value of the Lagrange multiplier. Its value depends on the starting point used in the steepest descent method.

If instead of the above constraint function $g(\mathbf{x})$, one uses

$$g(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - 1,$$

the converged solution must be scaled in order to get a normalized eigenvector.