Lab 2: Matrix Decompositions

Prof. Gaston Sanchez

Stat 154, Fall 2017

Introduction

In this lab you will be working with the functions `svd()` and `eigen()` to perform singular value decomposition, and eigenvalue decomposition, respectively.

In addition, you will have to write code in R to implement the Power Method, which is a basic procedure to obtain the dominant eigenvector (and related eigenvalue) of a square matrix.

The content of this lab will introduce you to a series of concepts and computations that will constantly emerge throughout the course. So it makes more sense to start learning first about these two fundamental decompositions (SVD & EVD) before tackling any of the statistical learning tasks (both supervised and unsupervised).

Data `USArrests`

In this lab we are going to use the data set `USArrests` that comes in R—see `?USArrests` for more information. This data set is about "Violent Crime Rates by US State", and contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

```r
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

The four variables are:

- `Murder`: numeric Murder arrests (per 100,000)
- `Assault`: numeric Assault arrests (per 100,000)
- `UrbanPop`: numeric Percent urban population
- `Rape`: numeric Rape arrests (per 100,000)

Singular Value Decomposition

Let $\mathbf{M}$ be an $n \times p$ matrix of full column-rank p (assume $n > p$). The singular value decomposition (SVD) of $\mathbf{M}$ is given by:

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$$

where:

- $\mathbf{U}$ is an $n \times p$ matrix of left singular vectors
- $\mathbf{D}$ is a $p \times p$ diagonal matrix of singular values
- $\mathbf{V}$ is a $p \times p$ matrix of right singular vectors

Function `svd()`

R provides the function `svd()` that allows you to compute the SVD of any rectangular matrix. Here's a basic example with `USArrests`:

```
SVD <- svd(USArrests)
```

The default output of `svd()` is a list with three elements:

- `u`: matrix of left singular vectors
- `v`: matrix of right singular vectors
- `d`: vector of singular values

You can use the arguments `nu` and `nv` to have more control over the behavior of `svd()`:

- `nu`: the number of left singular vectors to be computed.
- `nv`: the number of right singular vectors to be computed.

SVD on raw data

- Use `svd()` to compute the SVD of `USArrests`
- Take the output of `svd()` and create the matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$
- Confirm that the data of `USArrests` can be obtained as the product of: $\mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$

SVD and best Rank-one Approximations

As we saw in lecture, one of the most attractive uses of the SVD is that it allows you to decompose a matrix $\mathbf{M}$ as a sum of rank-one matrices of the form: $l_k \mathbf{u_k} \mathbf{v_k}^{\mathsf{T}}$. This is the result of the famous Eckart-Young-Mirsky theorem, which says that the best rank r approximation to $\mathbf{X}$ is given by:

$$\mathbf{X_r} = \sum_{k=1}^{r} l_k \mathbf{u_k} \mathbf{v_k}^\mathsf{T}$$

where l_k is the k-th singular vector, that is, the k-th diagonal element in $\mathbf{D}$.

Consequently, the best $r = 2$ rank approximation to $\mathbf{X}$ is:

$$\mathbf{X_2} = l_1 \mathbf{u_1} \mathbf{v_1}^\mathsf{T} + l_2 \mathbf{u_2} \mathbf{v_2}^\mathsf{T}$$

In other words, $\mathbf{u_1}$, and $\mathbf{u_2}$ form the best 2-dimensional approximation of the objects in $\mathbf{X}$ (living a p-dim space).

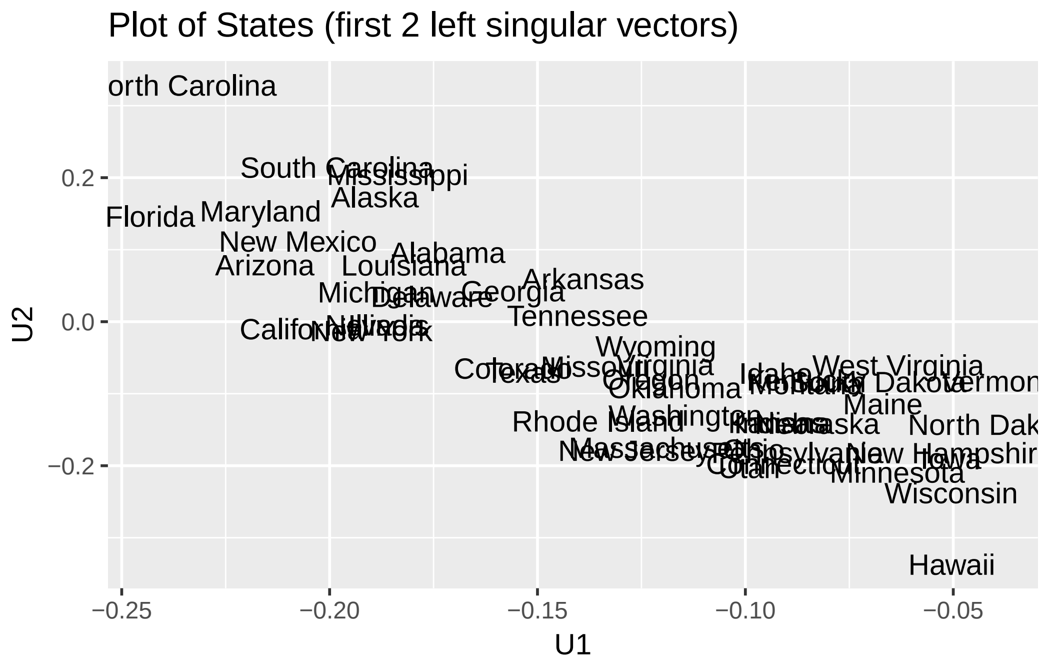
Your turn

- Confirm that the sum: $\sum_{k=1}^{4} l_h \mathbf{u_k} \mathbf{v_k}^\mathsf{T}$ equals `USArrests`
- Create a new variable (vector) `MA` by adding `Murder + Assault`
- Create a new data.frame `Arrests2` with the variables `Murder`, `Assault`, `UrbanPop`, `Rape`, and `MA`.
- Compute the SVD of `Arrests2`
- How does the singular values of `USArrests` compare to those of `Arrests2`?
- What is the rank of `Arrests2`?

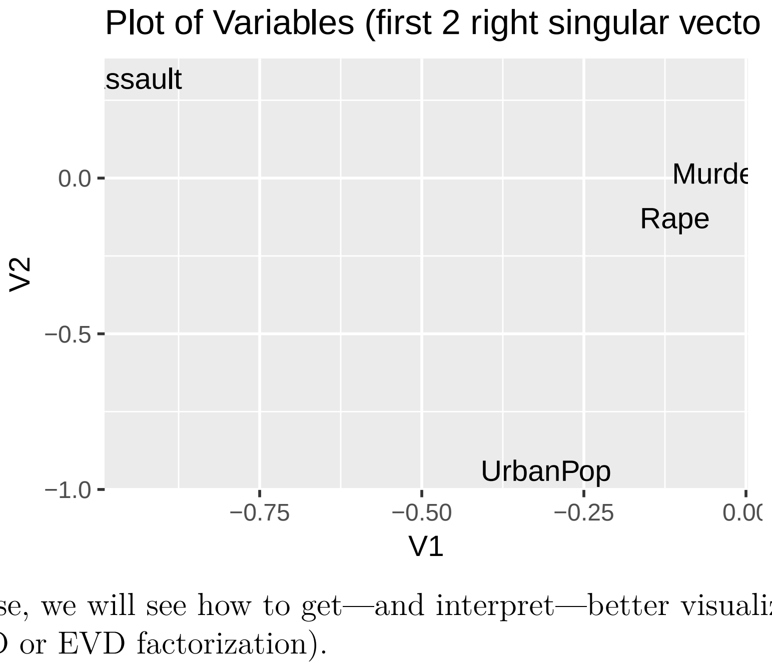
Lab continues on next page

Using SVD output to visualize data

Knowing that we can approximate `USArrests` with $l_1 \mathbf{u_1} \mathbf{v_1}^\mathsf{T} + l_2 \mathbf{u_2} \mathbf{v_2}^\mathsf{T}$, use $\mathbf{u_1}$ and $\mathbf{u_2}$ to visualize the States with a simple scatterplot. Create your own scatterplot like the one below:



In the same way, use the first two right singular vectors of $\mathbf{V}$ to graph a scatterplot (e.g. 2-dimensional representation) of the variables:



Later in the course, we will see how to get—and interpret—better visualizations (based on some kind of SVD or EVD factorization).

Eigenvalue Decomposition

R provides the function `eigen()` to compute the spectral decomposition—better known as *eigenvalue decomposition*—of a square matrix.

For multivariate analysis and statistical learning techniques, the typical square matrices are some variation of the sum-of-squares and cross-products: $\mathbf{X}^\mathsf{T}\mathbf{X}$ and $\mathbf{X}\mathbf{X}^\mathsf{T}$. For instance, assuming that variables in $\mathbf{X}$ are standardized (mean = 0, var = 1), the (sample) correlation matrix $\mathbf{R}$ is based on the product $\mathbf{X}^\mathsf{T}\mathbf{X}$, that is: $\mathbf{R} = \frac{1}{n-1}\mathbf{X}^\mathsf{T}\mathbf{X}$,.

```
R <- cor(USArrests)
```

The output of `eigen()` is a list of two elements, `values` and `vectors`.

```
# EVD of correlation matrix
evd <- eigen(R)
```

You can be more specific and use the argument `symmetric = TRUE` to indicate that the input matrix is symmetric (and don't let R guess it by itself):

```
evd <- eigen(R, symmetric = TRUE)
```

If you need only the eigenvalues you can use the parameter `only.values = TRUE`:

```
eigenvalues <- eigen(R, symmetric = TRUE, only.values = TRUE)
eigenvalues
```

```
## $values
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
##
## $vectors
## NULL
```

Inverse of covariance matrix

- Without using `scale()`, compute a matrix $\mathbf{X}$ as the mean-centered data of `USArrests`.
- Calculate the sum-of-squares and cross-products matrix $\mathbf{S} = \mathbf{X}^\mathsf{T}\mathbf{X}$
- $\mathbf{S}$ is proportional to the (sample) covariance matrix $\frac{1}{n-1}\mathbf{X}^\mathsf{T}\mathbf{X}$. Confirm that `cov(X)` is equal to $\frac{1}{n-1}\mathbf{X}^\mathsf{T}\mathbf{X}$.
- Use `solve()` to compute the inverse $\mathbf{S}^{-1}$
- Use `eigen()` to compute the EVD of $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\mathsf{T}$
- Confirm that $\mathbf{S}^{-1}$ can also be obtained as: $\mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^\mathsf{T}$

```
# mean-centered matrix X
means <- colMeans(USArrests)
X <- as.matrix(sweep(USArrests, 2, means, FUN = "-"))
```

```
# minor product moment XtX
S <- t(X) %*% X

# covariance matrix
n <- nrow(USArrests)
(1/(n-1)) * S
```

```
##              Murder  Assault   UrbanPop     Rape
## Murder    18.970465 291.0624   4.386204 22.99141
## Assault  291.062367 6945.1657 312.275102 519.26906
## UrbanPop   4.386204 312.2751 209.518776 55.76808
## Rape      22.991412 519.2691  55.768082 87.72916
```

```
cov(USArrests)
```

```
##              Murder  Assault   UrbanPop     Rape
## Murder    18.970465 291.0624   4.386204 22.99141
## Assault  291.062367 6945.1657 312.275102 519.26906
## UrbanPop   4.386204 312.2751 209.518776 55.76808
## Rape      22.991412 519.2691  55.768082 87.72916
```

```
# inverse of S via solve()
Sinv <- solve(S)

# inverse of S via EVD
S_evd <- eigen(S)
Vs <- S_evd$vectors
Ls <- diag(1/S_evd$values)
Vs %*%  Ls %*% t(Vs)
```

```
##               [,1]          [,2]          [,3]          [,4]
## [1,]  0.0032804923 -1.304887e-04  1.794220e-04 -2.014203e-04
## [2,] -0.0001304887  1.046419e-05 -6.597122e-06 -2.354634e-05
## [3,]  0.0001794220 -6.597122e-06  1.271111e-04 -8.877578e-05
## [4,] -0.0002014203 -2.354634e-05 -8.877578e-05  4.812179e-04
```

Power Method

The **Power Method** is an iterative procedure for approximating eigenvalues.

First assume that the matrix $\mathbf{A}$ has a dominant eigenvalue with corresponding dominant eigenvector. Then choose an initial approximation $\mathbf{w}_0$ (must be a non-zero vector) of one of the dominant eigenvectors. This choice is arbitrary (and in theory should work with almost any vector). Then, form the sequence $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k$, given by:

$$\begin{aligned}
\mathbf{w}_1 &= \mathbf{A}\mathbf{w}_0 \\
\mathbf{w}_2 = \mathbf{A}\mathbf{w}_1 &= \mathbf{A}^2\mathbf{w}_0 \\
\mathbf{w}_3 = \mathbf{A}\mathbf{w}_2 &= \mathbf{A}^3\mathbf{w}_0 \\
&\vdots \\
\mathbf{w}_k = \mathbf{A}\mathbf{w}_{k-1} &= \mathbf{A}^{k-1}\mathbf{w}_0
\end{aligned}$$

For large powers of k, and by **properly scaling** this sequence, you will see that you obtain a good approximation $\mathbf{w}_k$ of the dominant eigenvector of $\mathbf{A}$.

Here's an example. Consider the following matrix

$$\mathbf{A} = \begin{pmatrix} 2 & -12 \\ 1 & -5 \end{pmatrix}$$

Let's start with an initial nonzero vector $\mathbf{w}_0$

$$\mathbf{w}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

In R, we create $\mathbf{A}$ and $\mathbf{w}_0$

```
# square matrix
A <- matrix(c(2, 1, -12, -5), nrow = 2, ncol = 2)
```

```
# starting vector
w0 <- c(1, 1)
```

Let's see what happens in the first four iterations:

```
w_old <- w0

for (k in 1:4) {
  w_new <- A %*% w_old
  print(paste('iteration =', k))
  print(w_new)
  cat("\n")
  # update w_old
  w_old <- w_new
}
```

```
## [1] "iteration = 1"
##      [,1]
## [1,]  -10
## [2,]   -4
##
## [1] "iteration = 2"
##      [,1]
## [1,]   28
## [2,]   10
##
## [1] "iteration = 3"
##      [,1]
## [1,]  -64
## [2,]  -22
##
## [1] "iteration = 4"
##      [,1]
## [1,]  136
## [2,]   46
```

Note that the obtained vectors seem to be very different from one another. However, the difference is in the scale. This is why the part "properly scaling" is very important. How do you scale $\mathbf{w}_k$? Well, there are different options, but probably the simplest one is by using the L_{∞}-norm.

Reminder of L_p-norms:

- L_1-norm: $\sum_{i=1}^{n} |w_i|$
- L_2-norm: $\sqrt{\sum_{i=1}^{n} (w_i)^2}$
- L_{∞}-norm: $max\{|w_1|, \dots, |w_n|\}$
- L_p-norm: $(\sum_{i=1}^{n} |w_i|^p)^{1/p}$

Description of Power Method

Here is the procedure of the Power Method to find the largest eigenvalue and its corresponding eigenvector. Write code in R to implement such method.

1. Start with an arbitrary vector $\mathbf{w}_0$
2. Iteration for a series of steps $k = 0, 1, 2, \dots, n$ to form the series of $\mathbf{w}_k$ vectors:

   $$\mathbf{w}_{k+1} = \frac{\mathbf{A}\mathbf{w}_k}{s_{k+1}}$$

   where s_{k+1} is the entry of $\mathbf{A}\mathbf{w}_k$ which has the largest absolute value (this is actually a scaling operation dividing by the L_{∞}-norm).
3. When the scaling factors s_k are not changing much, s_{k+1} will be close to the largest eigenvalue of $\mathbf{A}$, and $\mathbf{w}_{k+1}$ will be close to the eigenvector associated to s_{k+1}.

4. You can also veirfy that s_{k+1} will be very close to the eigenvalue given by the Rayleigh quotient:

$$\lambda \approx \frac{\mathbf{w}_{k+1}^\mathsf{T} \mathbf{A} \mathbf{w}_{k+1}}{\mathbf{w}_{k+1}^\mathsf{T} \mathbf{w}_{k+1}}$$

Your turn: Implement the power method to find the largest eigenvalue of the following matrix:

$$\mathbf{A} = \begin{pmatrix} 5 & -14 & 11 \\ -4 & 4 & -4 \\ 3 & 6 & -3 \end{pmatrix}$$

Compare your results with those provided by `eigen()`. Keep in mind that the eigenvectors of `eigen()` have unit Euclidean norm (i.e. L_2-norm). Likewise, recall that the eigenvectors are only defined up to a constant: even when the length is specified they are still only defined up to a scalar.

Other scaling options

The scaling step in the power method:

$$\mathbf{w}_{k+1} = \frac{\mathbf{A}\mathbf{w}_k}{s_{k+1}}$$

involves choosing a value for s_{k+1}. One option for s_{k+1} is the entry of $\mathbf{A}\mathbf{w}_k$ with the largest absolute value (i.e. L_∞-norm). But you can actually use other scaling strategies. For instance, you can use the L_1-norm or the L_2-norm.

Modify your code to use any other L_p-norm in the power algorithm, and confirm that you get the same dominant eigenvectors and eigenvalues.

Deflation and more eigenvectors

In order to get more eigenvectors and eigenvalues, you need to apply the Power Method on the residual matrix obtained by **deflating A** with respect to the first eigenvector. This deflation operation is:

$$\mathbf{A}_1 = \mathbf{A} - \lambda_1 \mathbf{v}_1 \mathbf{v}_1^\mathsf{T}$$

where λ_1 is the first eigenvalue and $\mathbf{v}_1$ is the corresponding eigenvalue.

Deflate the matrix **A** and apply the power method on $\mathbf{A}_1$ to obtain more eigenvalues and eigenvectors.

```r
## @knitr matrix_A2
# square matrix
A <- cbind(
  c(5, 1),
  c(1, 5)
)


## @knitr power_method
# 1) dividing by L-max norm
set.seed(123)
w_old <- rnorm(2)

crit <- 1
iteration <- 1

while (crit > 1e-9 & iteration <= 50) {
  w_new <- A %*% w_old
  # scaling to L-max norm
  w_norm <- max(abs(w_new))
  w_new <- w_new / w_norm
  # check convergence
  w_diff <- abs(w_old) - abs(w_new)
  crit <- max(abs(w_diff))
  # print
  print(paste('iteration', iteration))
  print(paste('crit =', crit))
  print(w_new)
  cat("\n")
  # next iteration
  iteration <- iteration + 1
  w_old <- w_new
}
```

```
## [1] "iteration 1"
## [1] "crit = 0.439524353447787"
##            [,1]
## [1,] -1.0000000
## [2,] -0.5643303
##
## [1] "iteration 2"
## [1] "crit = 0.12248218028062"
##            [,1]
## [1,] -1.0000000
## [2,] -0.6868125
##
## [1] "iteration 3"
## [1] "crit = 0.0928971391706254"
##            [,1]
## [1,] -1.0000000
## [2,] -0.7797096
##
## [1] "iteration 4"
## [1] "crit = 0.0678326296513324"
##            [,1]
## [1,] -1.0000000
## [2,] -0.8475423
##
## [1] "iteration 5"
## [1] "crit = 0.0481693190480987"
##            [,1]
## [1,] -1.0000000
## [2,] -0.8957116
##
## [1] "iteration 6"
## [1] "crit = 0.0335329800806962"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9292446
##
## [1] "iteration 7"
## [1] "crit = 0.0230222518427066"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9522668
##
## [1] "iteration 8"
## [1] "crit = 0.015655872977308"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9679227
##
## [1] "iteration 9"
## [1] "crit = 0.0105774980352772"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9785002
##
## [1] "iteration 10"
## [1] "crit = 0.00711506318363009"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9856152
##
## [1] "iteration 11"
## [1] "crit = 0.00477187391452161"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9903871
##
## [1] "iteration 12"
## [1] "crit = 0.00319401180338807"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9935811
##
## [1] "iteration 13"
## [1] "crit = 0.00213504229144956"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9957162
##
## [1] "iteration 14"
## [1] "crit = 0.00142590394429798"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9971421
##
## [1] "iteration 15"
## [1] "crit = 0.000951735150284683"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9980938
##
## [1] "iteration 16"
## [1] "crit = 0.000634994202567407"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9987288
##
## [1] "iteration 17"
## [1] "crit = 0.000423553739399063"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9991524
##
## [1] "iteration 18"
## [1] "crit = 0.00028246890250605"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9994348
##
## [1] "iteration 19"
## [1] "crit = 0.000188356951687951"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9996232
##
## [1] "iteration 20"
## [1] "crit = 0.000125591018124593"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9997488
##
## [1] "iteration 21"
## [1] "crit = 8.37361102701273e-05"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9998325
##
## [1] "iteration 22"
## [1] "crit = 5.58279695207498e-05"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9998883
##
## [1] "iteration 23"
## [1] "crit = 3.72203780588665e-05"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999256
##
## [1] "iteration 24"
## [1] "crit = 2.48143550675151e-05"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999504
##
## [1] "iteration 25"
## [1] "crit = 1.65432454786174e-05"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999669
##
## [1] "iteration 26"
## [1] "crit = 1.10289823674892e-05"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999779
##
## [1] "iteration 27"
## [1] "crit = 7.35272249008379e-06"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999853
##
## [1] "iteration 28"
## [1] "crit = 4.90184502843771e-06"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999902
##
## [1] "iteration 29"
## [1] "crit = 3.26791003457672e-06"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999935
##
## [1] "iteration 30"
## [1] "crit = 2.17861262286068e-06"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999956
##
## [1] "iteration 31"
## [1] "crit = 1.45241105198313e-06"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999971
##
## [1] "iteration 32"
## [1] "crit = 9.68275206680858e-07"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999981
##
## [1] "iteration 33"
## [1] "crit = 6.45517325370548e-07"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999987
##
## [1] "iteration 34"
## [1] "crit = 4.30345114876829e-07"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999991
##
## [1] "iteration 35"
## [1] "crit = 2.86896846168894e-07"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999994
##
## [1] "iteration 36"
## [1] "crit = 1.91264609927799e-07"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999996
##
## [1] "iteration 37"
## [1] "crit = 1.27509760305955e-07"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999997
##
## [1] "iteration 38"
## [1] "crit = 8.50065158264357e-08"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999998
##
## [1] "iteration 39"
## [1] "crit = 5.667101454776e-08"
##            [,1]
## [1,] -1.0000000
## [2,] -0.9999999
##
## [1] "iteration 40"
## [1] "crit = 3.77806781415302e-08"
##             [,1]
## [1,] -1.0000000
## [2,] -0.9999999
##
## [1] "iteration 41"
## [1] "crit = 2.51871196121911e-08"
##             [,1]
## [1,] -1.0000000
## [2,] -0.9999999
##
## [1] "iteration 42"
## [1] "crit = 1.67914133708535e-08"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 43"
## [1] "crit = 1.11942758396211e-08"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 44"
## [1] "crit = 7.46285055974738e-09"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 45"
## [1] "crit = 4.97523378051312e-09"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 46"
## [1] "crit = 3.31682248333465e-09"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 47"
## [1] "crit = 2.21121498888976e-09"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 48"
## [1] "crit = 1.47414347395625e-09"
##      [,1]
## [1,]   -1
## [2,]   -1
##
## [1] "iteration 49"
## [1] "crit = 9.82762315970831e-10"
##      [,1]
## [1,]   -1
## [2,]   -1
```

```
## @knitr eigenvalue
# scaling factor(i.e. the norm) is close to eigenvalue
w_norm
```

```
## [1] 6
```

```
# eigenvalue with Rayleigh quotient
sum(w_new * (A %*% w_new)) / sum(w_new*w_new)
```

```
## [1] 6
```

```
w_1 <- w_new / sqrt(sum(w_new^2))
w_1
```

```
##            [,1]
## [1,] -0.7071068
## [2,] -0.7071068
```

```
## @knitr deflation
A1 <- A - w_norm * (w_1 %*% t(w_1))

set.seed(123)
w_old <- rnorm(2)

crit <- 1
iteration <- 1

while (crit >  1e-9 & iteration <= 50) {
  w_new <- A1 %*% w_old
  # scaling to L-max norm
  w_norm <- max(abs(w_new))
  w_new <- w_new / w_norm
  # check convergence
```

```r
  w_diff <- abs(w_old) - abs(w_new)
  crit <- max(abs(w_diff))
  # print
  print(paste('iteration', iteration))
  print(paste('crit =', crit))
  print(w_new)
  cat("\n")
  # next iteration
  iteration <- iteration + 1
  w_old <- w_new
}
```

```
## [1] "iteration 1"
## [1] "crit = 0.769822509936266"
##      [,1]
## [1,]   -1
## [2,]    1
##
## [1] "iteration 2"
## [1] "crit = 2.94828650382328e-09"
##      [,1]
## [1,]   -1
## [2,]    1
##
## [1] "iteration 3"
## [1] "crit = 0"
##      [,1]
## [1,]   -1
## [2,]    1
```

```r
w_new / sqrt(sum(w_new^2))
```

```
##            [,1]
## [1,] -0.7071068
## [2,]  0.7071068
```

```r
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 6 4
##
## $vectors
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```