12 Givens Rotations

Goal

Use elementary rotations to orthogonalize a set of given vectors.

Alert 12.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 12.2: Rotation

Let $I \in \mathbb{R}^{m\times m}$ be the identity matrix and fix two indices $i \neq j \in \{1, \ldots, m\}$ and some angle θ. Define

$$G^\top = G_{ij}^\top(\theta) = I - (1-\cos(\theta))\mathbf{e}_i\mathbf{e}_i^\top - (1-\cos(\theta))\mathbf{e}_j\mathbf{e}_j^\top - \sin(\theta)\mathbf{e}_i\mathbf{e}_j^\top + \sin(\theta)\mathbf{e}_j\mathbf{e}_i^\top$$

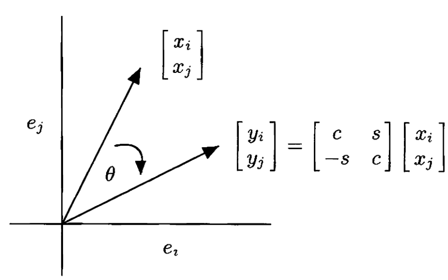
$$
= \begin{array}{cc}
 & \begin{array}{cccccccccccc} & & & i & & & & j & & & \end{array} \\
\begin{array}{c} \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \\ \end{array} &
\begin{bmatrix}
1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & \cos(\theta) & 0 & \cdots & 0 & -\sin(\theta) & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
\vdots & & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & \sin(\theta) & 0 & \cdots & 0 & \cos(\theta) & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}
\end{array}
$$

where $\mathbf{e}_i$ is the i-th canonical basis in $\mathbb{R}^m$, i.e., with a single 1 at the i-th entry. We easily verify that G is an orthogonal matrix: $G^\top G = I$, and $G^\top(\theta) = G(-\theta)$. In particular, G is invertible, and it is a rank-2 modification of the identity matrix.

The systematic use of rotations in numerical analysis was due to Givens (1958).

Givens, Wallace (1958). "Computation of Plain Unitary Rotations Transforming a General Matrix to Triangular Form". *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, no. 1, pp. 26–50.

Example 12.3: Geometric View



Exercise 12.4: Determinant

Prove that $\det(G) = 1$.

Remark 12.5: Structured Matrix-Vector Product

Multiplying a rotation with a vector can be done in linear time, instead of the usual quadratic time for a generic matrix:

$$[G^\top \mathbf{x}]_k = \begin{cases} x_k, & k \neq i, k \neq j \\ \cos(\theta)x_i - \sin(\theta)x_j, & k = i \\ \sin(\theta)x_i + \cos(\theta)x_j, & k = j \end{cases}.$$

Algorithm 12.6: Givens Orthogonal Triangularization

Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, can we find a rotation $G^\top = G_{ij}^\top(\theta)$ so that $\mathbf{y} = G^\top \mathbf{x}$? Since $G^\top$ only changes the i-th and j-th coordinate, and $G^\top$ is orthogonal, we obviously need $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2, x_k = y_k, k \neq i, k \neq j$. This, again, turns out to be sufficient. Indeed,

$$[G^\top \mathbf{x}]_k = \begin{cases} x_i\cos(\theta) - x_j\sin(\theta), & k = i \\ x_i\sin(\theta) + x_j\cos(\theta), & k = j \\ x_k, & \text{otherwise} \end{cases} \iff \begin{bmatrix} x_i & -x_j \\ x_j & x_i \end{bmatrix} \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} = \begin{bmatrix} y_i \\ y_j \end{bmatrix}$$

$$\iff c := \cos(\theta) = \frac{x_iy_i + x_jy_j}{x_i^2 + x_j^2}, \ s := \sin(\theta) = \frac{x_iy_j - x_jy_i}{x_i^2 + x_j^2},$$

provided that $x_i^2 + x_j^2 \neq 0$ (otherwise trivially we have $\cos(\theta) = 1$).

In particular, let $y_i = \sqrt{x_i^2 + x_j^2}, y_j = 0$, and $y_k = x_k$ otherwise, then we have

$$c = \cos(\theta) = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \ s = \sin(\theta) = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}.$$

Thus, by left-multiplying a rotation we can introduce one more zero in the vector $\mathbf{x}$. Repeating this $O(n^2)$ times gives us the Givens orthogonal triangularization algorithm.

In practice, we need only store one of $c = \cos(\theta)$ and $s = \sin(\theta)$ on the lower diagonal of A. The usual choice is the smaller one, since to recover the other one we need to compute $\sqrt{1 - x^2}$, which is numerically less accurate when x is close to 1. In the algorithm below we actually store $2/c$ if c is smaller and $s/2$ if s is smaller so that there is a unique encoding (Stewart 1976): if the storage is smaller than 1 then we know we stored $s/2$ while if the storage is bigger than 1 then we have stored $2/c$. The scaling factor 2 is chosen for convenience in a binary machine.

Given ρ we can easily recover (c, s), and we store Q in the factor form:

$$Q = G_{m-1,m;1} \cdots G_{1,2;1} \cdot G_{m-1,m;2} \cdots G_{2,3;2} \cdots G_{m-1,m;n\wedge(m-1)} \cdots G_{n\wedge(m-1),1+n\wedge(m-1);n\wedge(m-1)}, \quad (12.1)$$

where $G^{\top}_{i-1,i;j}$, $i = m, m-1, \ldots, j+1$, is the i-th rotation used on the j-column.

Algorithm: Givens QR

Input: $A \in \mathbb{R}^{m\times n}$
Output: $A = QR$, where $Q \in \mathbb{R}^{m\times n}$ orthogonal and $R \in \mathbb{R}^{n\times n}$ upper triangular.

for $j = 1, \ldots, n \wedge (m-1)$ **do**
  for $i = m, m-1, \ldots, j+1$ **do**
    $[c, s, \rho] =$ `givens`$(a_{i-1,j}, a_{ij})$ `// take a pair` $(i-1, i)$ `on` j`-th column and find rotation`
    $A_{(i-1):i,j:n} \leftarrow \begin{bmatrix} c & -s \\ s & c \end{bmatrix} A_{(i-1):i,j:n}$ `// annihilate` a_{ij}
    $a_{ij} \leftarrow \rho$ `// store rotation inplace`

Procedure $[c, s, \rho] =$ `givens`(a, b)
  if $b = 0$ **then**
    $c \leftarrow 1,\ s \leftarrow 0,\ \rho \leftarrow 0$ `// no need to rotate, pass`
  else if $a = 0$ **then**
    $c \leftarrow 0,\ s \leftarrow 1,\ \rho \leftarrow 1$ `// rotation does not need to be computed`
  else
    if $|b| > |a|$ **then**
      $\tau \leftarrow -a/b,\ s \leftarrow \frac{1}{\sqrt{1+\tau^2}},\ c \leftarrow s\tau,\ \rho \leftarrow 2/c$ `//` $|s| > |c| \implies |\rho| > 2\sqrt{2}$
    else
      $\tau \leftarrow -b/a,\ c \leftarrow \frac{1}{\sqrt{1+\tau^2}},\ s \leftarrow c\tau,\ \rho \leftarrow s/2$ `//` $|c| \geq |s| \implies |\rho| \leq \sqrt{2}/4$

Procedure $[c, s] =$ `givensInv`(ρ)
  if $\rho = 0$ **then**
    $c \leftarrow 1,\ s \leftarrow 0$
  else if $\rho = 1$ **then**
    $c \leftarrow 0,\ s \leftarrow 1$
  else if $|\rho| > 2$ **then**
    $c \leftarrow 2/\rho,\ s \leftarrow \sqrt{1-c^2}$
  else
    $s \leftarrow 2/\rho,\ c \leftarrow \sqrt{1-s^2}$

Stewart, G. W. (1976). "The economical storage of plane rotations". *Numerische Mathematik*, vol. 25, no. 2, pp. 137–138.

Remark 12.7: Complexity of Givens QR

The total number of FLOPs in Algorithm 12.6 is (assuming $m \geq n$):

$$\sum_{j=1}^{n} \sum_{i=j+1}^{m} 6(n-j+1) \sim \sum_{j=1}^{n} 6(m-j)(n-j) = 6mn^2 - 3mn^2 - 3n^3 + 2n^3 = 3mn^2 - n^3 = O(mn^2),$$

which is slower than the $2mn^2 - \frac{2}{3}n^3$ of Householder QR.

Example 12.8: Schematic Illustration

The main procedure in Algorithm 12.6 can be understood as follows:

$$
\underbrace{\begin{array}{cc}
 & \begin{array}{cccccccc} & & & i-1 & i & i+1 & & \end{array} \\
\begin{array}{c} \\ \\ \\ i-1 \\ i \\ i+1 \\ \\ \\ \end{array} &
\begin{bmatrix}
1 & & & & & & & \\
 & \ddots & & & & & & \\
 & & 1 & & & & & \\
 & & & c_{ij} & -s_{ij} & & & \\
 & & & s_{ij} & c_{ij} & & & \\
 & & & & & 1 & & \\
 & & & & & & \ddots & \\
 & & & & & & & 1
\end{bmatrix}
\end{array}}_{G^\top_{i-1,i;j}}
\cdot
\underbrace{\begin{array}{cc}
 & \begin{array}{ccccccc} & & j-1 & j & j+1 & & \end{array} \\
\begin{array}{c} \\ \\ \\ j \\ \\ i-1 \\ i \\ i+1 \\ \\ \\ \end{array} &
\begin{bmatrix}
* & \cdots & * & * & * & \cdots & * \\
 & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
 & & * & * & * & \cdots & * \\
 & & & * & * & \cdots & * \\
 & & & \vdots & \vdots & \ddots & \vdots \\
 & & & * & * & \cdots & * \\
 & & & * & * & \cdots & * \\
 & & & 0 & * & \cdots & * \\
 & & & \vdots & \vdots & \ddots & \vdots \\
 & & & 0 & * & * & *
\end{bmatrix}
\end{array}}_{A_{i,j}}
\to
\underbrace{\begin{array}{cc}
 & \begin{array}{ccccccc} & & j-1 & j & j+1 & & \end{array} \\
\begin{array}{c} \\ \\ \\ j \\ \\ i-1 \\ i \\ i+1 \\ \\ \\ \end{array} &
\begin{bmatrix}
* & \cdots & * & * & * & \cdots & * \\
 & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
 & & * & * & * & \cdots & * \\
 & & & * & * & \cdots & * \\
 & & & \vdots & \vdots & \ddots & \vdots \\
 & & & * & * & \cdots & * \\
 & & & 0 & * & \cdots & * \\
 & & & 0 & * & \cdots & * \\
 & & & \vdots & \vdots & \ddots & \vdots \\
 & & & 0 & * & * & *
\end{bmatrix}
\end{array}}_{A_{i-1,j}}
$$

Line 3 computes the rotation for the pair $(i-1, i)$ (highlighted in blue) at the j-th (outer) iteration. Note that due to the structure in $G^\top_{i-1,i;j}$, only the highlighted area (in blue) in $A_{i-1,j}$ gets updated. In other words, the structure in $G^\top_{i-1,i;j}$ makes sure we do not destroy any zeros introduced in previous iterations.

Algorithm 12.9: Explicit vs. Implicit

Note that we do not store each rotation G explicitly in Algorithm 12.6. For most applications, having the essential scalar ρ is enough, for we can perform the matrix-matrix multiplication $Q^\top C$, where Q is given in (12.1), efficiently:

Algorithm: Implicit Givens Matrix-Matrix Multiplication

Input: $A \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{m\times p}$
Output: inplace for $Q^\top C$

for $j = 1, \ldots, n \wedge (m-1)$ **do**
  for $i = m, m-1, \ldots, j+1$ **do**
    $[c, s] = \texttt{givensInv}(a_{i,j})$
    $C_{(i-1):i,:} \leftarrow \begin{bmatrix} c & -s \\ s & c \end{bmatrix} C_{(i-1):i,:}$ `//` $C \leftarrow G^\top_{i,i-1;j} C$

The above algorithm costs $3pn(2m-n)$. Similarly, we can efficiently compute QC as well.

Algorithm 12.10: Recovering Q

We can also explicitly recover the orthogonal matrix Q, by exploiting efficient matrix-matrix product:

Algorithm: Backward Recovery for Givens Orthogonal Matrix

Input: $A \in \mathbb{R}^{m\times n}$
Output: $Q \in \mathbb{R}^{m\times p}$

$Q \leftarrow I_m(:, 1:p)$ `// if only the first` p `columns need recovery`
for $j = n \wedge (m-1) \wedge p, \ldots, 2, 1$ **do**
  for $i = j+1, \ldots, m-1, m$ **do**
    $[c, s] = \texttt{givensInv}(a_{i,j})$
    $Q_{(i-1):i,j:p} \leftarrow \begin{bmatrix} c & s \\ -s & c \end{bmatrix} Q_{(i-1):i,j:p}$ `//` $Q \leftarrow G_{i,i-1;j} Q$

The above algorithm, known as backward accumulation, has complexity $6mnp - 3mn^2 - 3pn^2 + 2n^3$, assuming $m \geq p \geq n$. In particular, for $m \geq n = p$, recovering Q costs an additional $3mn^2 - n^3$. Again, we have exploited the sparisty pattern in I_m so that at the j-th iteration only the j-th to the p-th columns of Q need be updated (and become dense).

Algorithm 12.11: Hessenberg QR via Givens

Givens rotation can be used to introduce strategic and selective zeros. For example, when a matrix A is Hessenberg (i.e., $(1, n)$-banded), using rotations we can annihilate the sub-diagonal more efficiently:

Algorithm: Givens QR for Hessenberg matrices

Input: Hessenberg matrix $A \in \mathbb{R}^{m \times n}$
Output: inplace for QR decomposition

```
for j = 1, 2, ..., (n − 1) ∧ (m − 1) do
    [c, s, ρ] = givens(a_{jj}, a_{j+1,j})
    A_{j:(j+1),j:n} ← [c −s; s c] A_{j:(j+1),j:n}
    a_{j+1,j} ← ρ                                   // inplace store rotation
```

$$\mathbf{for}\ j = 1, 2, \ldots, (n-1) \wedge (m-1)\ \mathbf{do}$$
$$[c, s, \rho] = \texttt{givens}(a_{jj}, a_{j+1,j})$$
$$A_{j:(j+1),j:n} \leftarrow \begin{bmatrix} c & -s \\ s & c \end{bmatrix} A_{j:(j+1),j:n}$$
$$a_{j+1,j} \leftarrow \rho \qquad \texttt{// inplace store rotation}$$

The above algorithm costs only $3n^2$. If we use Householder and take sparsity into account, then the number of total FLOPs is $4n^2$.

Exercise 12.12: Givens QR for Tri-diagonal matrix

Let $A \in \mathbb{R}^{n \times n}$ be tri-diagonal. Design an efficient algorithm for the QR decomposition of A.

Exercise 12.13: Givens QR for Banded matrices

Adapt the Givens QR algorithm for a (p, q)-banded matrix.

Remark 12.14: Parallelism

Givens rotations can be easily parallelized: pairs that do not overlap can be updated in parallel (and the corresponding rotations commute), without interfering with each other. In other words, the pairs $(i_1, j_1; k_1)$ and $(i_2, j_2; k_2)$ can be updated in parallel if $\{i_1, i_2, j_1, j_2\}$ are distinct. In fact, using n processes (each corresponding to a column) we can perform Givens QR in $O((m + n)n)$ by arranging the pairs carefully:

steps \ processes	$j = 1$	$j = 2$	$\cdots$	$j = n-1$	$j = n$
1	$(m, m-1)$				
2	$(m-1, m-2)$				
3	$(m-2, m-3)$	$(m, m-1)$			
$\vdots$	$\vdots$	$\vdots$			
$m-2$	$(3, 2)$	$(5, 4)$	$\ddots$		
$m-1$	$(2, 1)$	$(4, 3)$	$\ddots$		
m		$(3, 2)$	$\ddots$		
$\vdots$			$\ddots$		
$2m-1$			$\ddots$	$(m, m-1)$	
$2n-2$			$\ddots$	$(m-1, m-2)$	
$2m-3$			$\ddots$	$(m-2, m-3)$	$(m, m-1)$
$\vdots$			$\ddots$	$\vdots$	$\vdots$
$m+n-4$			$\ddots$	$(n+1, n)$	$(n+3, n+2)$
$m+n-3$				$(n, n-1)$	$(n+2, n+1)$
$m+n-2$					$(n+1, n)$

At each step, if the pair $(i, i+1)$ is on process/column j, then the pair $(i+2, i+3)$ is on process $j+1$.

Hence, there is no conflict. Counting from top to bottom we observe that for $k = 1, \ldots n-1$, we have 3 steps with k processes concurrently running, hence there are $\frac{mn - \frac{n(n+1)}{2} - 3(n-1)}{n} = m - \frac{n+7}{2} + \frac{3}{n}$ steps where n processes are concurrently running.