Numerical Linear Algebra

- LU Decomposition.
- Partial and Full Pivoting.
- Cholesky Decomposition.

- Gaussian Elimination/LU decomposition transforms a full linear system into an upper-triangular one by applying simple linear transformations to the left.

Gaussian Elimination

- Gaussian Elimination/LU decomposition transforms a full linear system into an upper-triangular one by applying simple linear transformations to the left.
- It is similar to Householder triangularization – the difference is that the transformations applied are not unitary.

Gaussian Elimination

- Gaussian Elimination/LU decomposition transforms a full linear system into an upper-triangular one by applying simple linear transformations to the left.
- It is similar to Householder triangularization – the difference is that the transformations applied are not unitary.
- This is done by subtracting multiples of each row from subsequent rows!

Gaussian Elimination

- Gaussian Elimination/LU decomposition transforms a full linear system into an upper-triangular one by applying simple linear transformations to the left.
- It is similar to Householder triangularization – the difference is that the transformations applied are not unitary.
- This is done by subtracting multiples of each row from subsequent rows!
- This elimination method is equivalent to multiplying A by a sequence of lower-triangular matrices L_k on the left:

$$\underbrace{L_{m-1} \cdots L_2 L_1}_{L^{-1}} A = U$$

Gaussian Elimination

- Gaussian Elimination/LU decomposition transforms a full linear system into an upper-triangular one by applying simple linear transformations to the left.
- It is similar to Householder triangularization – the difference is that the transformations applied are not unitary.
- This is done by subtracting multiples of each row from subsequent rows!
- This elimination method is equivalent to multiplying A by a sequence of lower-triangular matrices L_k on the left:

$$\underbrace{L_{m-1} \cdots L_2 L_1}_{L^{-1}} A = U$$

- Setting $L = L_1^{-1} L_2^{-1} \cdots L_{m-1}^{-1}$ gives

$$A = LU$$

- The matrix L_k are chosen such that it introduces zeros below the diagonal in the kth column by subtracting multiples of row k from rows $k+1, \dots, m$.

- The matrix L_k are chosen such that it introduces zeros below the diagonal in the kth column by subtracting multiples of row k from rows $k+1, \ldots, m$.
- As the first $k-1$ entries are already zero, this operation does not destroy any zeroes previously obtained.

- The matrix L_k are chosen such that it introduces zeros below the diagonal in the kth column by subtracting multiples of row k from rows $k+1, \ldots, m$.
- As the first $k-1$ entries are already zero, this operation does not destroy any zeroes previously obtained.
- For example, in the 4×4 case, the zeroes are introduced in the following way:

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \xrightarrow{L_1} \begin{bmatrix} \times & \times & \times & \times \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix} \xrightarrow{L_2} \begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

$$\xrightarrow{L_3} \begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & \mathbf{0} & \boldsymbol{\times} \end{bmatrix}$$

LU decomposition

- Gram-Schmidt: $A = QR$ by triangular orthogonalization.
- Householder: $A = QR$ by orthogonal triangularization.
- Gaussian Elimination: $A = LU$ by triangular triangularization.

LU Decomposition

- Consider an $m \times m$ matrix. Suppose x_k denotes the kth column of the matrix beginning at step k. Then L_k must be chosen such that:

$$x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ x_{k+1,k} \\ \vdots \\ x_{mk} \end{bmatrix} \xrightarrow{L_k} L_k x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

LU Decomposition

- Consider an $m \times m$ matrix. Suppose x_k denotes the kth column of the matrix beginning at step k. Then L_k must be chosen such that:

$$x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ x_{k+1,k} \\ \vdots \\ x_{mk} \end{bmatrix} \xrightarrow{L_k} L_k x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- To do this, we subtract l_{jk} times row k from row j:

$$l_{jk} = \frac{x_{jk}}{x_{kk}} \quad (k < j \leq m)$$

LU Decomposition

- The matrix L_k takes the form:

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -l_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & -l_{mk} & & & 1 \end{bmatrix}$$

LU Decomposition

- The matrix L_k takes the form:

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -l_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & -l_{mk} & & & 1 \end{bmatrix}$$

- Define l_k as:

$$l_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{mk} \end{bmatrix}$$

- Then $L_k = I - l_k e_k^*$ where e_k is the column vector with 1 in position k and 0 otherwise.

- Then $L_k = I - l_k e_k^*$ where e_k is the column vector with 1 in position k and 0 otherwise.
- One can easily check that $e_k^* l_k = 0$.

- Then $L_k = I - l_k e_k^*$ where e_k is the column vector with 1 in position k and 0 otherwise.
- One can easily check that $e_k^* l_k = 0$.
- Therefore consider:

$$(I - l_k e_k^*)(I + l_k e_k^*) = I - l_k e_k^* l_k e_k^* = I$$

That is the inverse of L_k is $I + l_k e_k^*$.

- Then $L_k = I - l_k e_k^*$ where e_k is the column vector with 1 in position k and 0 otherwise.
- One can easily check that $e_k^* l_k = 0$.
- Therefore consider:

$$(I - l_k e_k^*)(I + l_k e_k^*) = I - l_k e_k^* l_k e_k^* = I$$

  That is the inverse of L_k is $I + l_k e_k^*$.
- Consider the product:

$$L_k^{-1} L_{k+1}^{-1} = (I + l_k e_k^*)(I + l_{k+1} e_{k+1}^*) = I + l_k e_k^* + l_{k+1} e_{k+1}^*$$

  Thus $L_k^{-1} L_{k+1}^{-1}$ is just a lower triangular matrix with entries of both L_k^{-1} and L_{k+1}^{-1} inserted in the usual places.

- As a result, we can write the full matrix L as:

$$L = L_1^{-1} L_2^{-1} \dots L_m^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{m1} & l_{m2} & \cdots & l_{m,m-1} & 1 \end{bmatrix}$$

Gaussian Elimination without Pivoting

The following algorithm computes the factor LU of A:

Algorithm 1 Gaussian Elimination without Pivoting

$U = A, L = I$
for $k = 1$ to $m - 1$ **do**
  for $j = k + 1$ to m **do**
    $l_{jk} = u_{jk}/u_{kk}$
    $u_{j,k:m} = u_{j,k:m} - l_{jk}u_{k,k:m}$
  end for
end for

The following algorithm computes the factor LU of A:

Algorithm 2 Gaussian Elimination without Pivoting

```
U = A, L = I
for k = 1 to m − 1 do
    for j = k + 1 to m do
        l_{jk} = u_{jk} / u_{kk}
        u_{j,k:m} = u_{j,k:m} − l_{jk} u_{k,k:m}
    end for
end for
```

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$

The following algorithm computes the factor LU of A:

Algorithm 3 Gaussian Elimination without Pivoting

```
U = A, L = I
for k = 1 to m − 1 do
    for j = k + 1 to m do
        l_{jk} = u_{jk} / u_{kk}
        u_{j,k:m} = u_{j,k:m} − l_{jk} u_{k,k:m}
    end for
end for
```

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$
- Work for (modified) Gram-Schmidt: $\sim 2mn^2$

Gaussian Elimination without Pivoting

The following algorithm computes the factor LU of A:

Algorithm 4 Gaussian Elimination without Pivoting

```
U = A, L = I
for k = 1 to m − 1 do
    for j = k + 1 to m do
        l_jk = u_jk / u_kk
        u_{j,k:m} = u_{j,k:m} − l_jk u_{k,k:m}
    end for
end for
```

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$
- Work for (modified) Gram-Schmidt: $\sim 2mn^2$
- Work for Gaussian elimination: $\sim \frac{2}{3}m^3$

Pivoting

- Gaussian elimination can become unstable/fail completely if the diagonal entries of the matrix A are very small/zero.

Pivoting

- Gaussian elimination can become unstable/fail completely if the diagonal entries of the matrix A are very small/zero.
- x_{kk} element plays an important role in Gaussian elimination and is called a pivot.

Pivoting

- Gaussian elimination can become unstable/fail completely if the diagonal entries of the matrix A are very small/zero.
- x_{kk} element plays an important role in Gaussian elimination and is called a pivot.
- If $x_{kk} = 0$ then we need a different (modified) algorithm. Even if $x_{kk} \neq 0$, but is small, there is a need for a more stable method.

Pivoting

- Gaussian elimination can become unstable/fail completely if the diagonal entries of the matrix A are very small/zero.
- x_{kk} element plays an important role in Gaussian elimination and is called a pivot.
- If $x_{kk} = 0$ then we need a different (modified) algorithm. Even if $x_{kk} \neq 0$, but is small, there is a need for a more stable method.
- In principle, there is no need to pick only x_{kk} as the pivot. In principle, we can use any element of $X_{k:m,k:m}$ as the pivot!

Pivoting

- Gaussian elimination can become unstable/fail completely if the diagonal entries of the matrix A are very small/zero.
- x_{kk} element plays an important role in Gaussian elimination and is called a pivot.
- If $x_{kk} = 0$ then we need a different (modified) algorithm. Even if $x_{kk} \neq 0$, but is small, there is a need for a more stable method.
- In principle, there is no need to pick only x_{kk} as the pivot. In principle, we can use any element of $X_{k:m,k:m}$ as the pivot!
- We can interchange columns/rows among themselves to bring a large number to the diagonal – rather than work with a smaller number.

Pivoting

- Gaussian elimination can become unstable/fail completely if the diagonal entries of the matrix A are very small/zero.
- x_{kk} element plays an important role in Gaussian elimination and is called a pivot.
- If $x_{kk} = 0$ then we need a different (modified) algorithm. Even if $x_{kk} \neq 0$, but is small, there is a need for a more stable method.
- In principle, there is no need to pick only x_{kk} as the pivot. In principle, we can use any element of $X_{k:m,k:m}$ as the pivot!
- We can interchange columns/rows among themselves to bring a large number to the diagonal – rather than work with a smaller number.
- This is crucial for stability of the algorithm.

- If any element of $X_{k:m,k:m}$ can be considered a pivot, then searching for the largest number will cost $\mathcal{O}(m-k)^2$ flops per step – overall cost for m steps $\mathcal{O}(m^3)$.

- If any element of $X_{k:m,k:m}$ can be considered a pivot, then searching for the largest number will cost $\mathcal{O}(m-k)^2$ flops per step – overall cost for m steps $\mathcal{O}(m^3)$.
- This strategy – is expensive an called complete pivoting.

Partial Pivoting

- If any element of $X_{k:m,k:m}$ can be considered a pivot, then searching for the largest number will cost $\mathcal{O}(m-k)^2$ flops per step – overall cost for m steps $\mathcal{O}(m^3)$.
- This strategy – is expensive an called complete pivoting.
- In practice, equally good pivots can be found by choosing the largest element from $(m-k+1)$ subdiagonal entries in column k. This can be achieved in $\mathcal{O}(m-k)$ operations and overall cost for finding the pivot is $\mathcal{O}(m^2)$.

Partial Pivoting

- If any element of $X_{k:m,k:m}$ can be considered a pivot, then searching for the largest number will cost $\mathcal{O}(m-k)^2$ flops per step – overall cost for m steps $\mathcal{O}(m^3)$.
- This strategy – is expensive an called complete pivoting.
- In practice, equally good pivots can be found by choosing the largest element from $(m-k+1)$ subdiagonal entries in column k. This can be achieved in $\mathcal{O}(m-k)$ operations and overall cost for finding the pivot is $\mathcal{O}(m^2)$.
- Then only rows are interchanged and it is called partial pivoting.

Partial Pivoting

- If any element of $X_{k:m,k:m}$ can be considered a pivot, then searching for the largest number will cost $\mathcal{O}(m-k)^2$ flops per step – overall cost for m steps $\mathcal{O}(m^3)$.
- This strategy – is expensive an called complete pivoting.
- In practice, equally good pivots can be found by choosing the largest element from $(m-k+1)$ subdiagonal entries in column k. This can be achieved in $\mathcal{O}(m-k)$ operations and overall cost for finding the pivot is $\mathcal{O}(m^2)$.
- Then only rows are interchanged and it is called partial pivoting.
- The interchange of rows can be represented by the application of the Permutation operator.

- This can be visualied as:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \boldsymbol{x_{ik}} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ & \boldsymbol{x_{ik}} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times & \times & \times \end{bmatrix}$$

$$\xrightarrow{L_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ & x_{ik} & \times & \times & \times \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

Partial Pivoting

- This can be visualied as:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \boldsymbol{x_{ik}} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ & \boldsymbol{x_{ik}} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times & \times & \times \end{bmatrix}$$

$$\xrightarrow{L_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ & x_{ik} & \times & \times & \times \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

- Then the upper triangular matrix can be:

$$L_{m-1}P_{m-1}\cdots L_2P_2L_2P_1A = U$$

Partial Pivoting

- Consider the following definition:

$$L'_k = P_{m-1} \dots P_{k+1} L_k P_{k+1}^{-1} \dots P_{m-1}^{-1}$$

- Consider the following definition:

$$L'_k = P_{m-1} \dots P_{k+1} L_k P_{k+1}^{-1} \dots P_{m-1}^{-1}$$

- Then:

$$\begin{aligned} U =& L_{m-1} P_{m-1} \cdots L_2 P_2 L_2 P_1 A \\ &= (L'_{m-1} \cdots L'_2 L'_1)(P_{m-1} \cdots P_2 P_1) A \end{aligned}$$

Partial Pivoting

- Consider the following definition:

$$L'_k = P_{m-1} \dots P_{k+1} L_k P_{k+1}^{-1} \dots P_{m-1}^{-1}$$

- Then:

$$\begin{aligned} U =& L_{m-1} P_{m-1} \cdots L_2 P_2 L_2 P_1 A \\ &= (L'_{m-1} \cdots L'_2 L'_1)(P_{m-1} \cdots P_2 P_1) A \end{aligned}$$

- Equivalent to solving $PA = LU$.

Gaussian Elimination with partial Pivoting

The following algorithm computes the factor LU of A:

Algorithm 5 Gaussian Elimination with Partial Pivoting

```
U = A, L = I, P = 1
for k = 1 to m − 1 do
    Select i ≥ k to maximize |u_{ik}|
    u_{k,k:m} ↔ u_{i,k:m}
    l_{k,k−1} ↔ l_{i,1:k−1}
    p_{k,:} ↔ p_{i,:}
    for j = k + 1 to m do
        l_{jk} = u_{jk}/u_{kk}
        u_{j,k:m} = u_{j,k:m} − l_{jk} u_{k,k:m}
    end for
end for
```

- Hermitian positive definite matrices can be decomposed into triangular factors twice as quickly as general matrices.

- Hermitian positive definite matrices can be decomposed into triangular factors twice as quickly as general matrices.
- The standard algorithm for this is the Cholesky factorization, which is a variant of Gaussian elimination that operates on left and right of the matrix at once.

- Hermitian positive definite matrices can be decomposed into triangular factors twice as quickly as general matrices.
- The standard algorithm for this is the Cholesky factorization, which is a variant of Gaussian elimination that operates on left and right of the matrix at once.
- For a complex matrix $A \in \mathbb{C}^{m \times m}$, Hermitian matrices are $A = A^*$.

- Hermitian positive definite matrices can be decomposed into triangular factors twice as quickly as general matrices.
- The standard algorithm for this is the Cholesky factorization, which is a variant of Gaussian elimination that operates on left and right of the matrix at once.
- For a complex matrix $A \in \mathbb{C}^{m \times m}$, Hermitian matrices are $A = A^*$.
- A Hermitian matrix is positive definite iff for any $x \in \mathbb{C}^m$, $x^* A x > 0$. The eigenvalues of Hermitian positive definite matrix are always positive and real.

Cholesky decomposition

- Consider what happens if we apply a single step of Gaussian elimination to a Hermitian matrix A with 1 in the upper left position:

$$A = \begin{bmatrix} 1 & w^* \\ w & K \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ w & I \end{bmatrix} \begin{bmatrix} 1 & w^* \\ 0 & K - ww^* \end{bmatrix}$$

- Consider what happens if we apply a single step of Gaussian elimination to a Hermitian matrix A with 1 in the upper left position:

$$A = \begin{bmatrix} 1 & w^* \\ w & K \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ w & I \end{bmatrix} \begin{bmatrix} 1 & w^* \\ 0 & K - ww^* \end{bmatrix}$$

- Gaussian elimination would now proceed with introducing zeros in the next column. However, in Cholesky factorization, they are introduced in the first row to keep the hermiticity of the matrix.

$$\begin{bmatrix} 1 & w^* \\ 0 & K - ww^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & K - ww^* \end{bmatrix} \begin{bmatrix} 1 & w^* \\ 0 & I \end{bmatrix}$$

Cholesky factorization

- Combining the two steps:

$$A = \begin{bmatrix} 1 & w^* \\ w & K \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ w & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^* \end{bmatrix} \begin{bmatrix} 1 & w^* \\ 0 & I \end{bmatrix}$$

- Combining the two steps:

$$A = \begin{bmatrix} 1 & w^* \\ w & K \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ w & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^* \end{bmatrix} \begin{bmatrix} 1 & w^* \\ 0 & I \end{bmatrix}$$

- The idea of Cholesky decomposition is to continue this process till the matrix is reduced to identity!

Cholesky factorization

- Combining the two steps:

$$A = \begin{bmatrix} 1 & w^* \\ w & K \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ w & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^* \end{bmatrix} \begin{bmatrix} 1 & w^* \\ 0 & I \end{bmatrix}$$

- The idea of Cholesky decomposition is to continue this process till the matrix is reduced to identity!
- In general, we need this to work for any $a_{11} > 0$. The generalization of this achieved by adjusting the algorithm and introducing $\alpha = \sqrt{a_{11}}$

$$A = \begin{bmatrix} a_{11} & w^* \\ w & K \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ w/\alpha & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^*/a_{11} \end{bmatrix} \begin{bmatrix} \alpha & w^*/\alpha \\ 0 & I \end{bmatrix}$$
$$= R_1^* A_1 R_1$$

Cholesky factorization

- If the upper left entry of the submatrix $K - ww^*/a_{11}$ is positive, the process can be continued further:

$$\begin{aligned} A =& \underbrace{R_1^* R_2^* \cdots R_m^*}_{R^*} \underbrace{R_m \cdots R_2 R_1}_{R} \\ =& R^* R \quad r_{jj} > 0 \end{aligned}$$

where R is upper triangular.

Cholesky factorization

- If the upper left entry of the submatrix $K - ww^*/a_{11}$ is positive, the process can be continued further:

$$
\begin{aligned}
A =& \underbrace{R_1^* R_2^* \cdots R_m^*}_{R^*} \underbrace{R_m \cdots R_2 R_1}_{R} \\
=& R^* R \quad r_{jj} > 0
\end{aligned}
$$

  where R is upper triangular.
- The only thing left hanging is that how do we know that the upper left entry of $K - ww^*/a_{11}$ is positive? It has to be because, $K - ww^*/a_{11}$ is positive definite as it is the principle submatrix of the positive definite matrix $R_1^{-*} A R_1^{-1}$.

Cholesky decomposition

The following algorithm computes the factor R^*R of complex Hermitian A:

Algorithm 6 Cholesky factorization

```
R = A
for k = 1 to m do
    for j = k + 1 to m do
        R_{j,j:m} = R_{j,j:m} - R_{k,j:m} conj(R_{kj}) / R_{kk}
    end for
    R_{k,k:m} = R_{k,k:m} / sqrt(R_{kk})
end for
```

$R = A$

for $k = 1$ to m **do**

  for $j = k+1$ to m **do**

    $R_{j,j:m} = R_{j,j:m} - R_{k,j:m}\overline{R_{kj}}/R_{kk}$

  end for

  $R_{k,k:m} = R_{k,k:m}/\sqrt{R_{kk}}$

end for

Cholesky decomposition

The following algorithm computes the factor R^*R of complex Hermitian A:

Algorithm 7 Cholesky factorization

$R = A$
for $k = 1$ to m **do**
  for $j = k + 1$ to m **do**
    $R_{j,j:m} = R_{j,j:m} - R_{k,j:m}\overline{R_{kj}}/R_{kk}$
  end for
  $R_{k,k:m} = R_{k,k:m}/\sqrt{R_{kk}}$
end for

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$

Cholesky decomposition

The following algorithm computes the factor R^*R of complex Hermitian A:

Algorithm 8 Cholesky factorization

```
1: R = A
2: for k = 1 to m do
3:     for j = k + 1 to m do
4:         R_{j,j:m} = R_{j,j:m} - R_{k,j:m} conj(R_{kj}) / R_{kk}
5:     end for
6:     R_{k,k:m} = R_{k,k:m} / sqrt(R_{kk})
7: end for
```

1: $R = A$
2: **for** $k = 1$ to m **do**
3: **for** $j = k+1$ to m **do**
4: $R_{j,j:m} = R_{j,j:m} - R_{k,j:m}\overline{R_{kj}}/R_{kk}$
5: **end for**
6: $R_{k,k:m} = R_{k,k:m}/\sqrt{R_{kk}}$
7: **end for**

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$
- Work for (modified) Gram-Schmidt: $\sim 2mn^2$

Cholesky decomposition

The following algorithm computes the factor R^*R of complex Hermitian A:

Algorithm 9 Cholesky factorization

```
R = A
for k = 1 to m do
    for j = k + 1 to m do
        R_{j,j:m} = R_{j,j:m} − R_{k,j:m} conj(R_{kj}) / R_{kk}
    end for
    R_{k,k:m} = R_{k,k:m} / sqrt(R_{kk})
end for
```

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$
- Work for (modified) Gram-Schmidt: $\sim 2mn^2$
- Work for Gaussian elimination: $\sim \frac{2}{3}m^3$

The following algorithm computes the factor R^*R of complex Hermitian A:

Algorithm 10 Cholesky factorization

1: $R = A$
2: **for** $k = 1$ to m **do**
3: **for** $j = k+1$ to m **do**
4: $R_{j,j:m} = R_{j,j:m} - R_{k,j:m}\overline{R_{kj}}/R_{kk}$
5: **end for**
6: $R_{k,k:m} = R_{k,k:m}/\sqrt{R_{kk}}$
7: **end for**

- Work for Householder orthogonalization $\sim 2mn^2 - \frac{2}{3}n^3$
- Work for (modified) Gram-Schmidt: $\sim 2mn^2$
- Work for Gaussian elimination: $\sim \frac{2}{3}m^3$
- Work for Cholesky factorization: $\sim \frac{1}{3}m^3$

- SVD Decomposition.
- Schur factorization.
- Eigenvalue finding.

- Singular Value Decomposition (SVD) is the generalization of the eigendecomposition of a positive semidefinite normal matrix to any $m \times n$ matrix.

- Singular Value Decomposition (SVD) is the generalization of the eigendecomposition of a positive semidefinite normal matrix to any $m \times n$ matrix.
- It is a factorization of a matrix M into:

$$M = U\Sigma V^*$$

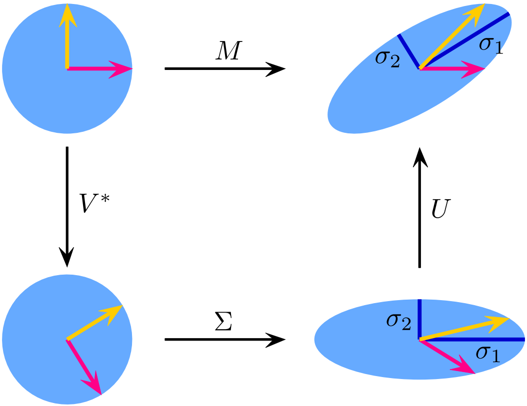
where U is $m \times m$ a unitary matrix, Σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal and V is a $n \times n$ unitary matrix.

- Singular Value Decomposition (SVD) is the generalization of the eigendecomposition of a positive semidefinite normal matrix to any $m \times n$ matrix.
- It is a factorization of a matrix M into:

$$M = U\Sigma V^*$$

  where U is $m \times m$ a unitary matrix, Σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal and V is a $n \times n$ unitary matrix.
- The diagonal entries σ_i of Σ are known as the singular values of M . The columns of U and the columns of V are called the left-singular vectors and right-singular vectors of M, respectively.

Physical meaning of SVD



$$M = U \cdot \Sigma \cdot V^*$$

Singular Value decomposition

- The left-singular vectors of M are a set of orthonormal eigenvectors of MM^*.

Singular Value decomposition

- The left-singular vectors of M are a set of orthonormal eigenvectors of MM^*.
- The right-singular vectors of M are a set of orthonormal eigenvectors of M^*M.

- The left-singular vectors of M are a set of orthonormal eigenvectors of MM^*.
- The right-singular vectors of M are a set of orthonormal eigenvectors of M^*M.
- The non-zero singular values of M (found on the diagonal entries of Σ) are the square roots of the non-zero eigenvalues of both M^*M and MM^*.

- The right-singular vectors corresponding to vanishing singular values of M span the null space of M

Rank, Null space and Range

- The right-singular vectors corresponding to vanishing singular values of M span the null space of M
- The left-singular vectors corresponding to the non-zero singular values of M span the range of M.

Rank, Null space and Range

- The right-singular vectors corresponding to vanishing singular values of M span the null space of M
- The left-singular vectors corresponding to the non-zero singular values of M span the range of M.
- The rank of M equals the number of non-zero singular values which is the same as the number of non-zero diagonal elements in Σ.

SVD and eigen decomposition

- Eigenvalue decomposition of a square matrix:

$$A = X\Lambda X^{-1}$$

where Λ is a diagonal matrix and X contains linearly independent eigenvectors of A.

SVD and eigen decomposition

- Eigenvalue decomposition of a square matrix:

$$A = X\Lambda X^{-1}$$

  where Λ is a diagonal matrix and X contains linearly independent eigenvectors of A.
- SVD is the generalization of eigen decomposition to rectangular matrices.

- Eigenvalue decomposition of a square matrix:

$$A = X \Lambda X^{-1}$$

  where Λ is a diagonal matrix and X contains linearly independent eigenvectors of A.
- SVD is the generalization of eigen decomposition to rectangular matrices.
- SVD uses two bases (left and right singular vectors) while eigenvalue decomposition uses only one (just the eigenvectors).

SVD and eigen decomposition

- Eigenvalue decomposition of a square matrix:

$$A = X\Lambda X^{-1}$$

  where Λ is a diagonal matrix and X contains linearly independent eigenvectors of A.
- SVD is the generalization of eigen decomposition to rectangular matrices.
- SVD uses two bases (left and right singular vectors) while eigenvalue decomposition uses only one (just the eigenvectors).
- In applications, SVD is relevant for problems involving the matrix itself where as eigen decomposition is useful to compute iterated forms of the matrix – such as matrix powers or exponentials etc.

Schur Factorization

- One final factorization:

$$A = QTQ^*$$

where Q is unitary and T is upper-triangular.

- One final factorization:

$$A = QTQ^*$$

  where Q is unitary and T is upper-triangular.
- Since A and T are similar, eigenvalues of A neccesarily appear on the diagonal of T.

Schur Factorization

- One final factorization:

$$A = QTQ^*$$

  where Q is unitary and T is upper-triangular.
- Since A and T are similar, eigenvalues of A neccesarily appear on the diagonal of T.
- Diagonalization algorithms use this factorization.

- Any eigenvalue solver has to be iterative!

- Any eigenvalue solver has to be iterative!
- Most of the general purpose eigenvalue algorithms proceed by computing the Schur factorization:

$$\underbrace{Q_j^* \cdots Q_2^* Q_1^*}_{Q^*} \, A \, \underbrace{Q_1 Q_2 \cdots Q_j}_{Q}$$

converges to an upper triangular matrix T as $j \to \infty$.

Two phases of eigenvalue computations

- Whether or not A is hermitian, the sequence is usually split into two phases – first a direct method is applied to produce a upper-Hessenberg matrix H, that is, a matrix with zeros below the first subdiagonal.

Two phases of eigenvalue computations

- Whether or not A is hermitian, the sequence is usually split into two phases – first a direct method is applied to produce a upper-Hessenberg matrix H, that is, a matrix with zeros below the first subdiagonal.
- In the second phase, an iteration is used to generate a formally infinite sequence of Hessenberg matrices that converge to a triangular form.

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{Phase 1} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

$$\xrightarrow{Phase 2} \begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{bmatrix}$$

- Use Householder reflectors to introduce zeros, but leave the first row as it is!

Reduction to Hessenberg/Tridiagonal form

- Use Householder reflectors to introduce zeros, but leave the first row as it is!
- Upon applying with Q_1 on the right, it will not destroy the zeroes that you have!

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{Q_1^*} \begin{bmatrix} \times & \times & \times & \times & \times \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

$$\xrightarrow{Q_1} \begin{bmatrix} \times & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \times & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ 0 & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ 0 & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ 0 & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

Householder reduction to Hessenberg Form

The following algorithm computes the Householder reduction of A to Hessenberg form:

Algorithm 1 Householder reduction to Hessenberg form

for $k = 1$ to $m - 2$ **do**
  $x = A_{k+1:m,k}$
  $v_k = \text{sign}(x_1)||x||_2 e_1 + x$
  $v_k = v_k / ||v_k||_2$
  $A_{k+1:m,k:n} = A_{k+1:m,k:n} - 2v_k(v_k^* A_{k+1:m,k:n})$
  $A_{1:m,k+1:n} = A_{1:m,k+1:n} - 2v_k(v_k^* A_{1:m,k+1:n})$
end for

Householder reduction to Tridiagonal form

- Reduces a symmetric/hermitian matrix to tridiagonal form.

Householder reduction to Tridiagonal form

- Reduces a symmetric/hermitian matrix to tridiagonal form.
- Work done $\sim \frac{4}{3}m^3 Flops$