

Linear Algebra

1.1 Invertability

Definition 1.1.1 (Singularity). An $n \times n$ square matrix \mathbf{A} is **non-singular** if it satisfies these conditions:

- i. \mathbf{A} has an inverse \mathbf{A}^{-1} such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} = \mathbf{A}^{-1}\mathbf{A}$.
- ii. $\det(\mathbf{A}) \neq 0$.
- iii. $\text{rank}(\mathbf{A}) = n$.
- iv. The null space $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$.

■

Discussion 1.1.1. If \mathbf{A} is non-singular, then the linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$ has a *unique* solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$. If \mathbf{A} is singular, there are two cases. If $\mathbf{y} \in \text{im}(\mathbf{A})$, then there are infinite many spaces, for if \mathbf{x}_0 is a particular solution and $\mathbf{z} \in \mathcal{N}(\mathbf{A})$, then every vector of the form $\mathbf{x}_0 + \alpha\mathbf{z}$ is a solution. Otherwise, if $\mathbf{y} \notin \text{im}(\mathbf{A})$, then no solutions exist. ♠

Discussion 1.1.2. The *Sherman-Morrison formula* gives the inverse for a rank-one modification of a matrix \mathbf{A} whose inverse is known:

$$(\mathbf{A} - \mathbf{u}\mathbf{v}')^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{u}(1 - \mathbf{v}'\mathbf{A}^{-1}\mathbf{u})^{-1}\mathbf{v}'\mathbf{A}^{-1}.$$

The *Woodbury formula* allows for a rank k modification:

$$(\mathbf{A} - \mathbf{U}\mathbf{V}')^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{U}(\mathbf{I} - \mathbf{V}'\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}'\mathbf{A}^{-1}.$$

♠

1.2 Norms

Definition 1.2.1 (Norm). The p -norm of a vector $\mathbf{x} \in \mathbb{R}^J$ is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{j=1}^J |x_j|^p \right)^{1/p}.$$

Important special cases include the L_1 (absolute value) norm, the L_2 (*Euclidean*) norm, and the L_∞ (*supremum*) norm:

$$\|\mathbf{x}\|_1 = \sum_{j=1}^J |x_j|, \quad \|\mathbf{x}\|_2 = \left(\sum_{j=1}^J |x_j|^2 \right)^{1/2}, \quad \|\mathbf{x}\|_\infty = \max_j |x_j|.$$

■

Example 1.2.1 (Matrix Norm). The *matrix norm* induced by a vector norm $\|\cdot\|_p$ is defined as:

$$\|\mathbf{A}\|_p = \sup_{\mathbf{x} \neq 0} \frac{\|\mathbf{Ax}\|_p}{\|\mathbf{x}\|_p} = \sup_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|_p.$$

For matrices, the 1-norm is the maximum absolute column sum:

$$\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^n |A_{ij}|,$$

and the sup-norm is the maximum absolute row sum:

$$\|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^J |A_{ij}|$$



Proposition 1.2.1. The L_2 norm of a matrix is $\|\mathbf{A}\|_2 = \sigma_{\max}(\mathbf{A})$.



Proof. By definition:

$$\|\mathbf{A}\|_2 = \sup_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|_2.$$

Taking the SVD:

$$\|\mathbf{A}\|_2 = \sup_{\|\mathbf{x}\|=1} \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}'\mathbf{x}\|_2.$$

For any vector \mathbf{y} :

$$\|\mathbf{U}\mathbf{y}\|_2^2 = (\mathbf{U}\mathbf{y})'(\mathbf{U}\mathbf{y}) = \mathbf{y}'\mathbf{U}'\mathbf{U}\mathbf{y} = \mathbf{y}'\mathbf{y} = \|\mathbf{y}\|_2^2,$$

since \mathbf{U} is an orthogonal matrix. Likewise, if $\|\mathbf{x}\| = 1$ then $\|\mathbf{V}'\mathbf{x}\| = \|\mathbf{x}\| = 1$. Thus:

$$\|\mathbf{A}\|_2 = \sup_{\|\mathbf{x}\|=1} \|\mathbf{U}\mathbf{\Sigma}\mathbf{V}'\mathbf{x}\|_2 = \sup_{\|\mathbf{y}\|=1} \|\mathbf{\Sigma}\mathbf{y}\|_2$$

Finally, since $\mathbf{\Sigma}$ is positive and diagonal:

$$\|\mathbf{A}\|_2 = \sup_{\|\mathbf{y}\|=1} \|\mathbf{\Sigma}\mathbf{y}\|_2 = \sigma_{\max}(\mathbf{A}).$$



Definition 1.2.2. The **condition number** of a matrix \mathbf{A} is defined as:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|.$$

If the matrix is singular, then $\text{cond}(\mathbf{A}) = \infty$.



Example 1.2.2 (Condition Number). Suppose $\mathbf{Ax} = \mathbf{y}$ and that \mathbf{y} is perturbed to $\mathbf{y} + \Delta\mathbf{y}$. Let $\mathbf{x} + \Delta\mathbf{x}$ denote the solution to the perturbed system of equations:

$$\mathbf{A}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{y} + \Delta\mathbf{y}.$$

Since $\mathbf{Ax} = \mathbf{y}$, $\Delta\mathbf{x}$ must satisfy:

$$\mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{y}, \quad \Delta\mathbf{x} = \mathbf{A}^{-1}\Delta\mathbf{y}$$

From the definition of the matrix norm:

$$\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|,$$

it follows immediately that for any particular \mathbf{x} :

$$\|\mathbf{Ax}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|.$$

Therefore:

$$\|\Delta\mathbf{x}\| = \|\mathbf{A}^{-1}\Delta\mathbf{y}\| \leq \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{y}\|$$

Dividing both sides by $\|\mathbf{x}\|$ and recalling that $\kappa(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$:

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(\mathbf{A})}{\|\mathbf{A}\| \cdot \|\mathbf{x}\|} \|\Delta\mathbf{y}\|.$$

Finally, since $\|\mathbf{A}\| \cdot \|\mathbf{x}\| \geq \|\mathbf{Ax}\| = \|\mathbf{y}\|$, upon taking reciprocals:

$$\frac{1}{\|\mathbf{A}\| \cdot \|\mathbf{x}\|} \leq \frac{1}{\|\mathbf{y}\|}.$$

Overall:

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(\mathbf{A})}{\|\mathbf{A}\| \cdot \|\mathbf{x}\|} \|\Delta\mathbf{y}\| \leq \kappa(\mathbf{A}) \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}\|}.$$

This equation says that the relative change in the solution \mathbf{x} due to a perturbation $\Delta\mathbf{y}$ in \mathbf{y} is bounded by the condition number $\kappa(\mathbf{A})$ times the relative change in \mathbf{y} .



1.3 Gaussian Elimination

Example 1.3.3 (*LU Decomposition*). Consider the system of equations $\mathbf{A}\boldsymbol{\beta} = \mathbf{y}$, where $\dim(\mathbf{A}) = n \times n$ and $\exists \mathbf{A}^{-1}$. The process of Gaussian elimination on a matrix \mathbf{A} is equivalent to left multiplication by a sequence of elementary elimination matrices, resulting in

an upper triangular matrix \mathbf{U} . In particular, the matrix \mathbf{M}_k that eliminates all entries of a column vector strictly below the k th position is:

$$\mathbf{M}_k = \mathbf{I} - \frac{1}{a_k} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1} \\ \vdots \\ a_n \end{pmatrix} \mathbf{e}'_k = \mathbf{I} - \mathbf{m}_k \mathbf{e}'_k.$$

The inverse matrix is $\mathbf{L}_k = \mathbf{M}_k^{-1} = \mathbf{I} + \mathbf{m}_k \mathbf{e}'_k$.

For $k_2 > k_1$, the operations may be chained:

$$\begin{aligned} \mathbf{M}_{k_1} \mathbf{M}_{k_2} &= \mathbf{I} - \mathbf{m}_{k_1} \mathbf{e}'_{k_1} - \mathbf{m}_{k_2} \mathbf{e}'_{k_2}, \\ \mathbf{L}_{k_1} \mathbf{L}_{k_2} &= \mathbf{I} + \mathbf{m}_{k_1} \mathbf{e}'_{k_1} + \mathbf{m}_{k_2} \mathbf{e}'_{k_2} \end{aligned}$$

Define the matrix $\mathbf{M} = \mathbf{M}_{n-1} \cdots \mathbf{M}_1$. The matrix $\mathbf{U} = \mathbf{M}\mathbf{A}$ obtained by sequentially eliminating columns $1, 2, \dots, n$ of matrix \mathbf{A} is upper triangular. The inverse:

$$\mathbf{L} = \mathbf{M}^{-1} = \mathbf{M}_1^{-1} \cdots \mathbf{M}_{n-1}^{-1} = \mathbf{L}_1 \cdots \mathbf{L}_{n-1}$$

is lower triangular. Note that the decomposition of \mathbf{L} is properly oriented for applying the chain rule, whereas the decomposition of \mathbf{M} is not.

Now, since $\mathbf{L}\mathbf{M} = \mathbf{I}$ and $\mathbf{M}\mathbf{A} = \mathbf{U}$, we have decomposed \mathbf{A} as $\mathbf{A} = \mathbf{L}\mathbf{U}$. This decomposition allows the problem $\mathbf{A}\boldsymbol{\beta} = \mathbf{y}$ to be efficiently solved for arbitrary \mathbf{y} :

- i. Write $\boldsymbol{\alpha} = \mathbf{U}\boldsymbol{\beta}$ and solve $\mathbf{L}\boldsymbol{\alpha} = \mathbf{y}$ by forward substitution.
- ii. Solve $\mathbf{U}\boldsymbol{\beta} = \boldsymbol{\alpha}$ by backward substitution.

Observe that step (i.) is equivalent to calculating $\mathbf{M}\mathbf{y}$ since the solution to $\mathbf{L}\boldsymbol{\alpha} = \mathbf{y}$ is $\boldsymbol{\alpha} = \mathbf{L}^{-1}\mathbf{y}$ and $\mathbf{M} = \mathbf{L}^{-1}$. In step (ii.), we find $\boldsymbol{\beta}$ such that $\mathbf{M}\mathbf{A}\boldsymbol{\beta} = \mathbf{U}\boldsymbol{\beta} = \boldsymbol{\alpha} = \mathbf{M}\mathbf{y}$. This demonstrates that $\boldsymbol{\beta}$ satisfying $\mathbf{U}\boldsymbol{\beta} = \boldsymbol{\alpha}$ is the same as that satisfying $\mathbf{A}\boldsymbol{\beta} = \mathbf{y}$.

If \mathbf{A}^{-1} exists, then \mathbf{A} admits an \mathbf{LU} factorization. However, direct application of Gaussian elimination will fail if the upper left element of the unreduced subsection of the matrix is zero. This element, a_k in the expression for \mathbf{M}_k is called the **pivot**, and *pivoting* refers to permuting the rows and columns of the unreduced sub-matrix such that the pivotal element is largest remaining entry. ♠

Example 1.3.4 (*Matrix Inversion*). Suppose the \mathbf{LU} factorization of an invertible matrix \mathbf{A} has been computed through the process described above. The inverse of \mathbf{A} is calculated by solving the n problems:

$$\mathbf{A}\mathbf{x}_k = \mathbf{e}_k,$$

where \mathbf{e}_k is the k th standard basis vector and $k \in \{1, \dots, n\}$. The matrix $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ obtained by horizontally concatenating the n coefficient vectors (\mathbf{x}_k) is the inverse of \mathbf{A} . ♠

Example 1.3.5 (*Row Echelon Form*). Forward Gaussian elimination procedures a matrix in *row echelon form*, where all rows consisting entirely of zeros are at the bottom, and the first non-zero element of each row, called in pivot, is always strictly to the right of the pivot for the row above. When a matrix is in row echelon form, the rank is the number of pivots. ♠

1.4 Iterative Linear Solvers

Example 1.4.6. The **Gauss-Seidel** method is an iterative approach to solving a linear system $\mathbf{Ax} = \mathbf{b}$. Starting from an initial guess \mathbf{x}_0 , the k th update for the i th component of \mathbf{x} is:

$$x_i^{(k+1)} = \frac{1}{A_{ii}} \left\{ b_i - \sum_{j < i} A_{ij} x_j^{(k+1)} - \sum_{j > i} A_{ij} x_j^{(k)} \right\}.$$

Write $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$ where \mathbf{D} is the diagonal, \mathbf{L} is the lower triangular sector, and \mathbf{U} is the upper triangular sector. Then, in matrix notation, the Gauss-Seidel update is:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}), \\ \mathbf{D}\mathbf{x}^{(k+1)} &= \mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}, \\ (\mathbf{D} + \mathbf{L})\mathbf{x}^{(k+1)} &= \mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}, \\ \mathbf{x}^{(k+1)} &= (\mathbf{D} + \mathbf{L})^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}). \end{aligned}$$

♠

Example 1.4.7. The **conjugate gradient** algorithm is an efficient method for solving $\mathbf{Ax} = \mathbf{b}$, when \mathbf{A} is symmetric and positive definite, based on the observation that $\mathbf{Ax} = \mathbf{b}$ is the minimum of the quadratic form:

$$\phi(\mathbf{x}) = \frac{1}{2} \mathbf{x}' \mathbf{A} \mathbf{x} - \mathbf{x}' \mathbf{b} + \mathbf{c}.$$

The negative gradient of ϕ is the residual vector:

$$-\frac{\partial\phi(\mathbf{x})}{\partial\mathbf{x}} = \mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{r}.$$

Consider an update of the form for minimizing ϕ :

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k,$$

where \mathbf{s}_k is the search direction. The optimal α_k satisfies:

$$0 = \frac{\partial\phi(\mathbf{x}_{k+1})}{\partial\alpha} = \frac{\partial\phi(\mathbf{x}_{k+1})}{\partial\mathbf{x}'_{k+1}} \frac{\partial\mathbf{x}_{k+1}}{\partial\alpha} = (\mathbf{A}\mathbf{x}_{k+1} - \mathbf{b})' \mathbf{r}_k = -\mathbf{r}'_{k+1} \mathbf{s}_k.$$

Because:

$$\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1} = \mathbf{b} - \mathbf{A}(\mathbf{x}_k + \alpha_k \mathbf{s}_k) = (\mathbf{b} - \mathbf{A}\mathbf{x}_k) - \alpha_k \mathbf{A}\mathbf{s}_k = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{s}_k,$$

the optimal α_k can be obtained analytically:

$$\begin{aligned} 0 &= (\mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{s}_k)' \mathbf{s}_k, \\ \alpha_k \mathbf{s}'_k \mathbf{A}\mathbf{s}_k &= \mathbf{r}'_k \mathbf{r}_k, \\ \alpha_k &= \frac{\mathbf{r}'_k \mathbf{r}_k}{\mathbf{s}'_k \mathbf{A}\mathbf{s}_k}. \end{aligned}$$

The search directions are chosen to be *conjugate* or *orthogonal* w.r.t. \mathbf{A} . That is:

$$\mathbf{s}'_k \mathbf{A}\mathbf{s}_l = 0 \text{ for } k \neq l.$$

The $(k+1)$ st search direction may be formed from the corresponding residual \mathbf{r}_{k+1} via Gram-Schmidt:

$$\mathbf{s}_{k+1} = \mathbf{r}_{k+1} - \sum_{j=1}^k \frac{\mathbf{r}'_{k+1} \mathbf{A}\mathbf{s}_j}{\mathbf{s}'_j \mathbf{A}\mathbf{s}_j} \mathbf{s}_j.$$

It can be shown that:

$$\mathbf{s}_{k+1} = \mathbf{r}_{k+1} + \frac{\mathbf{r}'_{k+1} \mathbf{r}_{k+1}}{\mathbf{r}'_k \mathbf{r}_k} \mathbf{s}_k,$$

leading to algorithm (1).



Algorithm 1 Conjugate Gradient Solver

Require: Initial guess \mathbf{x}_0 .

```

1:  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ .
2:  $\mathbf{s}_0 = \mathbf{r}_0$  ▷ Initial search direction.
3: for  $k = 1, \dots$  do
4:    $\alpha_k = \mathbf{r}'_k \mathbf{r}_k / \mathbf{s}'_k \mathbf{A} \mathbf{s}_k$ . ▷ Step-size.
5:    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$ .
6:    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{s}_k$ . ▷ Update residual.
7:    $\beta_{k+1} = \mathbf{r}'_{k+1} \mathbf{r}_{k+1} / \mathbf{r}'_k \mathbf{r}_k$ .
8:    $\mathbf{s}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{s}_k$ . ▷ New search direction.
9: end for
```

1.5 Eigenvalues

Definition 1.5.1. An **eigenvector** of a square matrix \mathbf{A} is a vector \mathbf{v} such that:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}.$$

The eigenvector is a direction of the input space along which the linear transform \mathbf{A} applies only scaling. The scalar λ is the *eigenvalue* corresponding to the eigenvector \mathbf{v} . The set $\lambda(\mathbf{A})$ of eigenvalues of \mathbf{A} is called its *spectrum*, and the maximum modulus of the eigenvalues $\max\{|\lambda| : \lambda \in \lambda(\mathbf{A})\}$ is the *spectral radius* of \mathbf{A} . ■

Definition 1.5.2. The **characteristic equation** of an $n \times n$ matrix \mathbf{A} is the n th degree polynomial in λ :

$$\phi(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I})$$

The *roots* of the characteristic equation, i.e. the solutions to $\phi(\lambda) \stackrel{\text{Set}}{=} 0$, are the eigenvalues of \mathbf{A} . ■

Example 1.5.8 (Multiplicity). By the fundamental theorem of algebra, an n th degree polynomial always has n roots in the complex plane, counting repeated roots. Since the eigenvalues of \mathbf{A} are the roots of its characteristic equation $\phi(\lambda)$, a square $n \times n$ matrix has a full set of n eigenvalues. The *algebraic multiplicity* of an eigenvalue is the multiplicity of that root in the characteristic polynomial. The *geometric multiplicity* of an eigenvalue is the number of linearly independent eigenvectors corresponding to that eigenvalue. An $n \times n$ matrix with fewer than n eigenvalues is described as *defective*. An $n \times n$ matrix \mathbf{A} with a full complement of n linearly independent eigenvectors $\mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_n)$ is said to be *diagonalizable* since:

$$\mathbf{A}\mathbf{U} = \mathbf{U}\mathbf{\Lambda}, \qquad \mathbf{U}^{-1}\mathbf{A}\mathbf{U} = \mathbf{\Lambda},$$

where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n)$. ♠

Definition 1.5.3. The **eigenspace** \mathcal{S}_λ corresponding to an eigenvalue is the linear subspace spanned by the linearly independent eigenvectors associated with λ . An eigenspace is *invariant* with respect to \mathbf{A} , meaning that if $\mathbf{x} \in \mathcal{S}_\lambda$, then $\mathbf{Ax} \in \mathcal{S}_\lambda$. ■

Example 1.5.9 (*Transformations*). If $\mathbf{Av} = \lambda\mathbf{v}$, then for $\sigma \in \mathbb{R}$:

$$(\mathbf{A} - \sigma\mathbf{I})\mathbf{v} = \lambda\mathbf{v} - \sigma\mathbf{v} = (\lambda - \sigma)\mathbf{v}.$$

That is, $\lambda - \sigma$ is an eigenvalue of the shifted matrix $\mathbf{A} - \sigma\mathbf{I}$.

If \mathbf{A} is non-singular, then:

$$\mathbf{v} = \lambda\mathbf{A}^{-1}\mathbf{v}, \quad \mathbf{A}^{-1}\mathbf{v} = \lambda^{-1}\mathbf{v}.$$

Thus, λ^{-1} is an eigenvalue of \mathbf{A}^{-1} .

If $\mathbf{Av} = \lambda\mathbf{v}$, then:

$$\mathbf{A}^2\mathbf{v} = \mathbf{A}(\lambda\mathbf{v}) = \lambda^2\mathbf{v},$$

and in general λ^k is an eigenvalue of \mathbf{A}^k for $k \in \mathbb{N}$.

Finally, recall that matrices \mathbf{A} and \mathbf{B} are *similar* if there exists a change of basis matrix \mathbf{S} such that $\mathbf{B} = \mathbf{S}^{-1}\mathbf{AS}$. If \mathbf{v} is an eigenvector of \mathbf{B} , then:

$$\lambda\mathbf{v} = \mathbf{Bv} = \mathbf{S}^{-1}\mathbf{ASv}, \quad \mathbf{A}(\mathbf{Sv}) = \lambda(\mathbf{Sv}).$$

Conclude that similar matrices have the same eigenvalues, and that the eigenvectors of \mathbf{A} are the eigenvectors of \mathbf{B} transformed by \mathbf{S} . ♠

Theorem 1.5.1 (Schur Decomposition). If \mathbf{A} is an $n \times n$ matrix, there exists a real, orthogonal matrix \mathbf{V} such that $\mathbf{A} = \mathbf{VUV}'$, where \mathbf{U} is upper block triangular. The diagonal entries of \mathbf{U} are 1×1 or 2×2 matrices corresponding to real and complex conjugate eigenvalue pairs, respectively. If complex entries are admitted, $\mathbf{A} = \mathbf{VUV}^*$ where \mathbf{V} is unitary and \mathbf{U} is upper triangular. □

Corollary 1.5.1. Every square matrix is *similar to* an upper triangular matrix. ♣

Example 1.5.10 (*Computation*). Suppose \mathbf{A} is an $n \times n$ matrix with a full complement of eigenvectors, and that the eigenvalues are sorted in descending order by magnitude. Let \mathbf{x}_0 denote an arbitrary input vector, and $\mathbf{U}\boldsymbol{\alpha}$ the representation of \mathbf{x}_0 with respect to the eigenbasis. Then:

$$\mathbf{A}^k \mathbf{x}_0 = \mathbf{A}^k \sum_{i=1}^n \alpha_i \mathbf{u}_i = \sum_{i=1}^n \alpha_i \mathbf{A}^k \mathbf{u}_i = \sum_{i=1}^n \alpha_i \lambda_i^k \mathbf{u}_i = \lambda_1^k \left\{ \alpha_1 \mathbf{u}_1 + \sum_{i=2}^n \alpha_i (\lambda_i / \lambda_1)^k \mathbf{u}_i \right\}.$$

Algorithm 2 Normalized Power Iteration

Require: Initialization vector \mathbf{x}_0 .

```

1: for  $k = 1, \dots$  do
2:    $\mathbf{y}_k = \mathbf{A}\mathbf{x}_{k-1}$ .
3:    $\mathbf{x}_k = \mathbf{y}_k / \|\mathbf{y}_k\|_\infty$ 
4: end for

```

As $k \rightarrow \infty$, $(\lambda_i/\lambda_1)^k \rightarrow 0$, and $\mathbf{A}^k \mathbf{x}_0 \rightarrow \lambda_1^k \mathbf{u}_1$. The **power iteration** seeks the leading eigenvalue of \mathbf{A} via:

The *inverse iteration* seeks the trailing eigenvalue of \mathbf{A} based on the observation that the trailing eigenvalue of \mathbf{A} is the leading eigenvalue of \mathbf{A}^{-1} . To implement the inverse iteration, step (2) of Algorithm 3 is replaced by $\mathbf{A}\mathbf{y}_k = \mathbf{x}_{k-1}$; implicitly $\mathbf{y}_k = \mathbf{A}^{-1}\mathbf{x}_{k-1}$.

An acceleration to the power iteration is based on the *Rayleigh quotient*. If \mathbf{x} is an eigenvalue of \mathbf{A} , then $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ and $\mathbf{x}'\mathbf{A}\mathbf{x} = \lambda\mathbf{x}'\mathbf{x}$, such that:

$$\lambda = \frac{\mathbf{x}'\mathbf{A}\mathbf{x}}{\mathbf{x}'\mathbf{x}}.$$

In the iteration, the leading eigenvalue is first approximated as σ_k via the Rayleigh quotient. \mathbf{A} is then shifted such that $\lambda_1 - \sigma_k$ is near zero. \mathbf{u}_1 becomes the trailing eigenvector of \mathbf{A} , and the leading eigenvector of \mathbf{A}^{-1} .

Algorithm 3 Rayleigh Quotient Iteration

Require: Initialization vector \mathbf{x}_0 .

```

1: for  $k = 1, \dots$  do
2:   Set  $\sigma_k = \mathbf{x}'_{k-1}\mathbf{A}\mathbf{x}_{k-1}/(\mathbf{x}'_{k-1}\mathbf{x}_{k-1})$ .
3:   Solve  $(\mathbf{A} - \sigma_k\mathbf{I})\mathbf{y}_k = \mathbf{x}_{k-1}$ . ▷ Inverse iteration.
4:    $\mathbf{x}_k = \mathbf{y}_k / \|\mathbf{y}_k\|_\infty$ 
5: end for

```

Having calculated the leading eigenvalue λ_1 of \mathbf{A} , successive eigenvalues may be obtained via *deflation*. Suppose \mathbf{u}_1 is the leading eigenvector and \mathbf{v}_1 is any vector such that $\mathbf{v}'_1\mathbf{u}_1 = \lambda_1$, then the deflated matrix $\mathbf{A} - \mathbf{u}_1\mathbf{v}'_1$ has eigenvalues $0, \lambda_2, \dots, \lambda_n$. In the case that \mathbf{A} is symmetric, the standard choice for \mathbf{v}_1 is $\lambda_1\mathbf{u}_1$, where \mathbf{u}_1 has been normalized such that $\mathbf{u}'_1\mathbf{u}_1 = 1$.



1.6 QR Decomposition

Definition 1.6.1. A **Householder** transformation is a matrix of the form:

$$\mathbf{H} = \mathbf{I} - 2 \frac{\mathbf{v}\mathbf{v}'}{\mathbf{v}'\mathbf{v}},$$

for $\mathbf{v} \neq \mathbf{0}$, and has the property:

$$\mathbf{H} = \mathbf{H}' = \mathbf{H}^{-1}.$$

That is, \mathbf{H} is both *symmetric* and *orthogonal*. ■

Example 1.6.11. Suppose we seek an orthogonal transformation of a vector \mathbf{a} such that all components of \mathbf{a} except the first are annihilated. That is:

$$\mathbf{H}\mathbf{a} = \alpha\mathbf{e}_1.$$

The vector \mathbf{v} must satisfy:

$$\mathbf{a} - 2\mathbf{v} \frac{\mathbf{v}'\mathbf{a}}{\mathbf{v}'\mathbf{v}} = \mathbf{H}\mathbf{a} = \alpha\mathbf{e}_1,$$

or upon rearranging:

$$\mathbf{v} = (\mathbf{a} - \alpha\mathbf{e}_1) \frac{\mathbf{v}'\mathbf{v}}{2\mathbf{v}'\mathbf{a}}.$$

We may further note that the scale factor $\mathbf{v}'\mathbf{v}/(2\mathbf{v}'\mathbf{a})$ divides out when forming the Householder matrix, and simply take $\mathbf{v} = \mathbf{a} - \alpha\mathbf{e}_1$. To preserve the norm and avoid cancellation, α is typically chosen as $\alpha = -\text{sign}(a_1)\|\mathbf{a}\|$.

More generally, to construct a Householder transformation that annihilates the last $n - k$ elements of \mathbf{a} , we use the prototype vector:

$$\mathbf{v} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_k \\ \dots \\ a_n \end{pmatrix} - \alpha\mathbf{e}_k.$$

The scalar α is take as $-\sigma(a_k)\|\mathbf{a}_{(k)}\|$, where $\mathbf{a}_{(k)}$ is the vector formed from \mathbf{a} by replacing the first $k - 1$ elements with zeros. ♠

Example 1.6.12. To interpret the Householder transformation geometrically, recall that:

$$\mathbf{P} = \mathbf{v}(\mathbf{v}'\mathbf{v})^{-1}\mathbf{v}'$$

is orthogonal projection onto the span of \mathbf{v} , and that:

$$\mathbf{P}^\perp = \mathbf{I} - \mathbf{v}(\mathbf{v}'\mathbf{v})^{-1}\mathbf{v}'$$

is projection onto the orthogonal complement. The overall transformation:

$$\mathbf{H} = \mathbf{P}^\perp - \mathbf{P} = \mathbf{I} - 2\mathbf{P} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}'}{\mathbf{v}'\mathbf{v}},$$

may be viewed as projection onto $\text{im}(\mathbf{v})^\perp$ followed by removal of an additional component in $\text{im}(\mathbf{v})$. The first projection \mathbf{P}^\perp moves \mathbf{a} onto the hyperplane with surface normal \mathbf{v} . Further subtracting $\mathbf{P}\mathbf{a}$ moves $\mathbf{P}^\perp\mathbf{a}$ from the hyperplane onto the coordinate axes. ♠

Example 1.6.13. Suppose \mathbf{A} is an $n \times p$ matrix, $n > p$, with full column rank. Sequential application of Householder transformations to a matrix $\mathbf{A}_{n \times p}$, from left to right, results in a matrix of the form:

$$\mathbf{H}_n \mathbf{H}_{n-1} \cdots \mathbf{H}_1 \mathbf{A} = \begin{pmatrix} \mathbf{R}_{p \times p} \\ \mathbf{0}_{(n-p) \times p} \end{pmatrix}.$$

Multiplying by the inverses (transposes) of the Householder transformations gives:

$$\mathbf{A} = \mathbf{H}_1' \mathbf{H}_2' \cdots \mathbf{H}_n' \begin{pmatrix} \mathbf{R}_{p \times p} \\ \mathbf{0}_{(n-p) \times p} \end{pmatrix} \equiv \mathbf{Q} \begin{pmatrix} \mathbf{R}_{p \times p} \\ \mathbf{0}_{(n-p) \times p} \end{pmatrix},$$

where $\mathbf{Q} = \mathbf{H}_1' \mathbf{H}_2' \cdots \mathbf{H}_n'$ is an $n \times n$ orthogonal matrix and \mathbf{R} is a $p \times p$ upper triangular matrix. The first p columns of \mathbf{Q} form an orthogonal basis for $\text{im}(\mathbf{A})$, while the last $n - p$ form a basis for the orthogonal complement. ♠

Example 1.6.14. The QR iteration can be used to obtain the eigenvalues of a matrix \mathbf{A} through successive *similarity transformations*:

$$\mathbf{A}_{k+1} = \mathbf{R}_k \mathbf{Q}_k = \mathbf{Q}_k' \mathbf{Q}_k \mathbf{R}_k \mathbf{Q}_k = \mathbf{Q}_k' \mathbf{A}_k \mathbf{Q}_k = \mathbf{Q}_k^{-1} \mathbf{A}_k \mathbf{Q}_k.$$

The algorithm converges to a triangular matrix. Because similarity transforms preserve eigenvalues, the eigenvalues of the original matrix are given by the diagonal of the limiting matrix. The algorithm is numerically stable because each similarity transform is implemented by an orthogonal matrix. If the original matrix \mathbf{A} is symmetric, then the limiting matrix is diagonal. ♠

1.7 Singular Value Decomposition

Definition 1.7.1. For an $n \times p$ matrix \mathbf{A} , the singular value decomposition (SVD) is:

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}'$$

Algorithm 4 QR Iteration

```

1: Let  $\mathbf{A}_0 = \mathbf{A}$ .
2: for  $k = 1, \dots$  do
3:   Decompose  $\mathbf{A}_k = \mathbf{Q}_k \mathbf{R}_k$ .
4:   Update  $\mathbf{A}_{k+1} = \mathbf{R}_k \mathbf{Q}_k$ .
5: end for

```

where \mathbf{U} is an $n \times n$ orthogonal matrix, \mathbf{V} is a $p \times p$ orthogonal matrix, and $\mathbf{\Sigma}$ is an $n \times p$ “diagonal” matrix with:

$$\Sigma_{ij} = \begin{cases} \sigma_i \geq 0 & i = j \\ 0, & i \neq j. \end{cases}$$

■

Example 1.7.15 (Applications of SVD). Consider an $n \times p$ matrix \mathbf{A} . If $n > p$ and \mathbf{A} has full column rank, then the SVD is expressible as:

$$\mathbf{A} = \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_0 \end{pmatrix} \begin{pmatrix} \mathbf{\Sigma}_1 \\ \mathbf{0}_{(n-p) \times p} \end{pmatrix} \mathbf{V}'$$

where $\mathbf{\Sigma}_1$ has all non-zero singular values along the diagonal, \mathbf{U}_1 is an orthogonal basis for $\text{im}(\mathbf{A})$ and \mathbf{U}_0 is an orthogonal basis for the orthogonal complement $\text{im}(\mathbf{A})^\perp$.

If $n < p$ and \mathbf{A} has full row rank, then the SVD is expressible as:

$$\mathbf{A} = \mathbf{U} \begin{pmatrix} \mathbf{\Sigma}_1 & \mathbf{0}_{n \times (p-n)} \end{pmatrix} \begin{pmatrix} \mathbf{V}'_1 \\ \mathbf{V}'_0 \end{pmatrix}$$

where $\mathbf{\Sigma}_1$ has all non-zero singular values along the diagonal, \mathbf{V}_0 is an orthogonal basis for $\ker(\mathbf{A})$, and \mathbf{V}_1 is an orthogonal basis for the orthogonal complement.

The **Moore-Penrose** inverse is a matrix \mathbf{A}^+ that in general has the properties:

$$\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}, \quad \mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+.$$

When $n > p$ and \mathbf{A} has full column rank, then:

$$\mathbf{A}^+ = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}', \quad \mathbf{A}^+\mathbf{A} = \mathbf{I}.$$

When $n < p$ and \mathbf{A} has full row rank, then:

$$\mathbf{A}^+ = \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1}, \quad \mathbf{A}\mathbf{A}^+ = \mathbf{I}.$$

In general, the Moore-Penrose inverse is given by:

$$\mathbf{A}^+ = \mathbf{U}\mathbf{\Sigma}^+\mathbf{V}'$$

where:

$$\Sigma^+ = \begin{cases} 1/\Sigma_{ij} & \Sigma_{ij} \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The SVD additionally has these properties:

- The rank of \mathbf{A} is the number of non-zero singular values of \mathbf{A} .
- The Euclidean norm of a matrix is $\|\mathbf{A}\|_2 = \sigma_{\max}(\mathbf{A})$ and the condition number is:

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}.$$

- A rank k approximation to \mathbf{A} is given by:

$$\hat{\mathbf{A}}_k = \sum_{j=1}^k \sigma_j \mathbf{u}_j \otimes \mathbf{v}_j.$$



Example 1.7.16 (Computation of the SVD). Numerically, the SVD is obtained by solving two eigenvalue problems. To find \mathbf{V} , we consider the matrix:

$$\mathbf{A}'\mathbf{A} = \mathbf{V}\Sigma'\mathbf{U}'\mathbf{U}\Sigma\mathbf{V} = \mathbf{V}\Sigma'\Sigma\mathbf{V}'.$$

Note that Σ is not symmetric when \mathbf{A} is not square, however the product $\Sigma'\Sigma$ is a square diagonal matrix with elements $\sigma_i^2 \geq 0$ along the diagonal. Similarly, to find \mathbf{U} , we consider the matrix:

$$\mathbf{A}\mathbf{A}' = \mathbf{U}\Sigma\mathbf{V}'\mathbf{V}\Sigma'\mathbf{U} = \mathbf{U}\Sigma\Sigma'\mathbf{U}'$$

Here $\Sigma\Sigma'$ is a square diagonal matrix with $\sigma_i^2 \geq 0$ along the diagonal. Because both $\mathbf{A}'\mathbf{A}$ and $\mathbf{A}\mathbf{A}'$ are symmetric, the spectral theorem guarantees the existence of each eigenvalue decomposition.



1.8 Normal Equations

Discussion 1.8.1. Consider the problem of finding \mathbf{x} such that $\mathbf{A}\mathbf{x}$ approximates \mathbf{y} . If $\mathbf{y} \notin \text{im}(\mathbf{A})$, then the equation $\mathbf{A}\mathbf{x} \stackrel{\text{Set}}{=} \mathbf{y}$ has no solution. However, provided \mathbf{A} has full column rank, we may obtain a *unique* least squares solution $\hat{\mathbf{x}}$ such that $\mathbf{A}\hat{\mathbf{x}}$

approximates, though does not equal, \mathbf{y} . The least squares solution $\hat{\mathbf{x}}$ is *defined* as the solution to the optimization problem:

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2.$$

The objective function is a quadratic form:

$$Q(\mathbf{x}) = \mathbf{x}'\mathbf{A}'\mathbf{A}\mathbf{x} + \mathbf{y}'\mathbf{y} - 2\mathbf{y}'\mathbf{A}\mathbf{x}.$$

Taking the gradient, $\hat{\mathbf{x}}$ must satisfy the first order condition:

$$\frac{\partial Q}{\partial \mathbf{x}} = 2\mathbf{A}'\mathbf{A}\mathbf{x} - 2\mathbf{y}'\mathbf{A} \stackrel{\text{Set}}{=} \mathbf{0}.$$

Rearranging gives the **normal equations**:

$$\mathbf{A}'\mathbf{A}\mathbf{x} = \mathbf{A}'\mathbf{y}.$$

If \mathbf{A} has full column rank, then the normal equations admit a *unique* solution $\hat{\mathbf{x}} = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'\mathbf{y}$ even if $\mathbf{y} \notin \text{im}(\mathbf{A})$. Notice that the matrix multiplying \mathbf{y} in the least squares solution is the Moore-Penrose left inverse:

$$\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{y}.$$

If \mathbf{A} is rank deficient, then $\mathbf{A}'\mathbf{A}$ is only positive semi-definite. A Moore-Penrose inverse of $\mathbf{A}'\mathbf{A}$ exists and we may write:

$$\tilde{\mathbf{x}} = (\mathbf{A}'\mathbf{A})^+\mathbf{A}'\mathbf{y}.$$

However, the solution is no longer unique; there are infinitely many solutions of the form $\tilde{\mathbf{x}} + \mathbf{k}$, where \mathbf{k} is a vector in the null space of $\mathbf{A}'\mathbf{A}$. ♠

Example 1.8.17. Let $\mathbf{A} = \mathbf{Q}_1\mathbf{R}$ denote the QR decomposition of $\mathbf{A}_{n \times p}$, where \mathbf{Q}_1 is an $n \times p$ matrix with orthonormal columns, and \mathbf{R} is a $p \times p$ upper triangular matrix. Let \mathbf{Q}_0 denote an $n \times (n - p)$ basis for the orthogonal complement of $\text{im}(\mathbf{Q}_1)$, such that $(\mathbf{Q}_1, \mathbf{Q}_0)$ is a complete orthogonal basis for $\mathbb{R}^{n \times n}$. Consider finding coefficients \mathbf{x} to minimize the least squares criterion $\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$. Because \mathbf{Q} is an orthogonal matrix:

$$\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{Q}(\mathbf{y} - \mathbf{A}\mathbf{x})\|_2^2 = \left\| \begin{pmatrix} \mathbf{Q}'_1 \\ \mathbf{Q}'_0 \end{pmatrix} (\mathbf{y} - \mathbf{Q}_1\mathbf{R}\mathbf{x}) \right\|_2^2 = \left\| \begin{pmatrix} \mathbf{Q}'_1\mathbf{y} - \mathbf{R}\mathbf{x} \\ \mathbf{Q}'_0\mathbf{y} \end{pmatrix} \right\|_2^2.$$

As only \mathbf{x} is variable, the sum of squares is minimized by taking $\mathbf{Q}'_1\mathbf{y} = \mathbf{R}\mathbf{x}$, and since \mathbf{R} is upper triangular, \mathbf{x} is easily obtained via back-substitution. Note that explicitly finding \mathbf{Q}_0 is unnecessary. ♠

1.8.1 Regularization

Example 1.8.18. Consider finding \mathbf{x} to minimize the residual $\mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{x}$, where $\mathbf{A}_{n \times p}$ is potentially rank-deficient. In Tikhonov regularization, the problem is augmented to:

$$\begin{pmatrix} \mathbf{A} \\ \mathbf{\Gamma} \end{pmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix},$$

where $\mathbf{\Gamma}$ is an $m \times p$ matrix, for example $\mathbf{\Gamma} = \epsilon \mathbf{I}_{p \times p}$. The least squares solution to the augmented problem satisfies the normal equations:

$$(\mathbf{A}'\mathbf{A} + \mathbf{\Gamma}'\mathbf{\Gamma})\mathbf{x} = \mathbf{A}'\mathbf{y}.$$

In the case $\mathbf{\Gamma} = \epsilon \mathbf{I}$, addition of $\epsilon^2 \mathbf{I}$ to $\mathbf{A}'\mathbf{A}$ ensures that $(\mathbf{A}'\mathbf{A} + \epsilon^2 \mathbf{I})$ is invertible even if \mathbf{A} is rank-deficient. While ensuring a solution exists, the added equations $\epsilon \mathbf{I}\mathbf{x} = \mathbf{0}$ bias the elements of the solution vector $\hat{\mathbf{x}} = (\mathbf{A}'\mathbf{A} + \epsilon^2 \mathbf{I})^{-1} \mathbf{A}'\mathbf{y}$ towards 0. ♠

Nonlinear Equations

Discussion 2.0.1. We seek \mathbf{x} such that $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ satisfies:

$$\mathbf{f}(\mathbf{x}) = \mathbf{y}.$$

Or equivalently $\mathbf{y} - \mathbf{f}(\mathbf{x}) = \mathbf{r}(\mathbf{x}) = \mathbf{0}$. If $n > m$, then the system is overdetermined, and an approximate solution may be sought via nonlinear least squares. If $n < m$, then the system is underdetermined, and the problem may be recast as constrained optimization. For $n = m$, the problem of finding \mathbf{x} such that $\mathbf{r}(\mathbf{x}) = \mathbf{0}$ is that of **root finding**. ♠

2.1 Existence and Uniqueness

Remark 2.1.1. Unlike linear systems, which always have 0, 1, or infinitely many solutions, nonlinear equations can have any number of solutions. ♦

Example 2.1.1 (Bracket Criterion). In one-dimension, the intermediate value theorem guarantees that if f is continuous on a closed interval $[a, b]$ and the signs of $f(a)$ and $f(b)$ differ, then there exists $c \in (a, b)$ such that $f(c) = 0$. Now, for an *upward crossing*, $f(a) \leq 0$ and $f(b) \geq 0$, which is equivalent to $(x - z)f(x) \geq 0$ for $x \in \{a, b\}$ and $z \in (a, b)$. If f has an upward crossing on $[a, b]$, then there exists $c \in (a, b)$ such that $f(c) = 0$. In n dimensions, if $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is continuous on the closure \bar{S} of an open bounded set S and $(\mathbf{x} - \mathbf{z})'f(\mathbf{x}) \geq 0$ for $\mathbf{x} \in \partial S$ any $\mathbf{z} \in S$, then there exists $\mathbf{c} \in S$ such that $\mathbf{f}(\mathbf{c}) = \mathbf{0}$. ♠

Definition 2.1.1. The **Jacobian** of $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is defined as:

$$\mathcal{J}_{ij} = \frac{\partial f_i}{\partial x_j}.$$

Notice that the components of f form the rows, and the components of \mathbf{x} , the input, form the columns. The resulting matrix is $m \times n$. ■

Theorem 2.1.1 (Inverse Function Theorem). If f is continuously differentiable and the Jacobian \mathcal{J} is *non-singular* at \mathbf{x}^* , then there exists a neighborhood of $f(\mathbf{x}^*)$ in which f is invertible. In particular, for \mathbf{y} belonging to the neighborhood of $f(\mathbf{x}^*)$, a solution to $f(\mathbf{x}) = \mathbf{y}$ exists. □

Definition 2.1.2. A mapping $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is **contractive** if there exists a constant $\lambda \in (0, 1)$ such that:

$$\|g(\mathbf{x}) - g(\mathbf{y})\| \leq \lambda \|\mathbf{x} - \mathbf{y}\|.$$

That is, the distance between the images is less than the distance between the inputs. A point \mathbf{x}^* is a **fixed point** of g if:

$$g(\mathbf{x}^*) = \mathbf{x}^*.$$

■

Theorem 2.1.2 (Contraction Mapping Theorem). If g is a *contractive mapping* on a closed set $\bar{S} \subseteq \mathbb{R}^n$ and $g(\bar{S}) \subseteq \bar{S}$, then g has a unique fixed point in \bar{S} . This implies that the function $f(\mathbf{x}) = g(\mathbf{x}) - \mathbf{x}$ has a zero in \bar{S} . □

2.2 Solving Nonlinear Equations

Example 2.2.2 (Newton's Iteration). Consider $f : \mathbb{R} \rightarrow \mathbb{R}$. Recall that the tangent line to f at x_0 is $y - f(x_0) = f'(x_0) \cdot (x - x_0)$. The tangent line intersects $y = 0$ at:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)},$$

which suggests the update rule:

$$x_{k+1} = x_k - f(x_k)/f'(x_k)$$

In multiple dimensions, the Jacobian $\mathcal{J}_k = \mathcal{J}(\mathbf{x}_k)$ replaces $f'(x_k)$ in the Newton algorithm, and the update state \mathbf{s}_k satisfies $\mathcal{J}_k \mathbf{s}_k = -f(\mathbf{x}_k)$. Although not required in principle, a step size α_k is sometimes applied to the update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$. The step size may be determined by line search:

$$\alpha_k = \arg \min_{\alpha} \|f(\mathbf{x}_k + \alpha \mathbf{s}_k)\|_2^2.$$

Algorithm 5 Newton's Method**Require:** Initial guess \mathbf{x}_0 .

- 1: **for** $k = 1, \dots$ **do**
- 2: Solve $\mathcal{J}_k \mathbf{s}_k = -f(\mathbf{x}_k)$ for the search direction \mathbf{s}_k .
- 3: Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$.
- 4: **end for**

The *secant method* replaces $f'(x_k)$ by the difference quotient:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

In multiple dimensions, the approximation \mathcal{Q}_k to the Jacobian used by the secant method satisfies:

$$\mathcal{Q}_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)$$

Algorithm 6 Secant Method**Require:** Initial guess \mathbf{x}_0 .**Require:** Initial Jacobian \mathcal{Q}_0 .

- 1: **for** $k = 1, \dots$ **do**
- 2: Solve $\mathcal{Q}_k \mathbf{s}_k = -f(\mathbf{x}_k)$ for the search direction \mathbf{s}_k .
- 3: Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.
- 4: Let $\mathbf{y}_k = f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)$.
- 5: Update:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{(\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k) \mathbf{s}_k'}{\mathbf{s}_k' \mathbf{s}_k}.$$

- 6: **end for**



Example 2.2.3 (*Brent's Method*). Given evaluations of $f : \mathbb{R} \rightarrow \mathbb{R}$ at three points:

$$y_1 = f(x_1), \quad y_2 = f(x_2), \quad y_3 = f(x_3),$$

we may fit an inverse parabola for x as a function of y . The corresponding Lagrange polynomial is:

$$x = g(y) = \frac{(y - y_2)(y - y_3)}{(y_1 - y_2)(y_1 - y_3)} x_1 + \frac{(y - y_1)(y - y_3)}{(y_2 - y_1)(y_2 - y_3)} x_2 + \frac{(y - y_1)(y - y_2)}{(y_3 - y_1)(y_3 - y_2)} x_3.$$

The interpolated root is:

$$x = \frac{y_2 y_3}{(y_1 - y_2)(y_1 - y_3)} x_1 + \frac{y_1 y_3}{(y_2 - y_1)(y_2 - y_3)} x_2 + \frac{y_1 y_2}{(y_3 - y_1)(y_3 - y_2)} x_3 \\ \equiv q(x_1, x_2, x_3; y_1, y_2, y_3).$$

Algorithm 7 Inverse Quadratic Iteration

Require: Three initial evaluation points x_0, x_1, x_2 .

- 1: Let $y_1 = f(x_1)$, $y_2 = f(x_2)$, $y_3 = f(x_3)$.
 - 2: **for** $k = 3, \dots$ **do**
 - 3: $x_{k+1} = q(x_{k-2}, x_{k-1}, x_k; y_{k-2}, y_{k-1}, y_k)$.
 - 4: $y_{k+1} = f(x_{k+1})$.
 - 5: **end for**
-

In Brent's method, an update is attempted based on either the secant method or inverse quadratic interpolation, which have super-linear convergence. However, if the proposed update falls outside the current root-bracketing interval, the bisection method is used as a fallback.



Optimization

3.1 Existence and Uniqueness

Definition 3.1.1. Let $f : \mathbb{R}^p \rightarrow \mathbb{R}$ denote an objective function and $S \subseteq \mathbb{R}^p$ a subset of *feasible solutions*. The subset is defined by a set of *constraints*. An **optimization problem** consists of finding the *minimizer*:

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in S} f(\mathbf{x}).$$



Discussion 3.1.1 (*Coercive Functions*). The extreme value theorem from analysis ensures that a function f defined on a closed and bounded set S has a global minimum. If S is not closed or is unbounded, then existence of a global minimum is not guaranteed. A function f defined on an unbounded set S is *coercive* if $\lim_{\|\mathbf{x}\| \rightarrow \infty} f(\mathbf{x}) = \infty$. A coercive function on a closed but unbounded set has a global minimum.



Definition 3.1.2. The **level sets** of a function $f : S \subseteq \mathbb{R}^m \rightarrow \mathbb{R}$ are of the form:

$$\mathcal{L}(\gamma) = \{\mathbf{x} \in S : f(\mathbf{x}) = \gamma\}.$$

The **sublevel sets** take the form:

$$\mathcal{L}^-(\gamma) = \{\mathbf{x} \in S : f(\mathbf{x}) \leq \gamma\}.$$

■

Definition 3.1.3. A set S is **convex** if for any \mathbf{x}_0 and \mathbf{x}_1 in S , the line segment:

$$\{\alpha\mathbf{x}_0 + (1 - \alpha)\mathbf{x}_1 : 0 \leq \alpha \leq 1\}$$

lies entirely in S . A function f is **convex** if the graph of the function lies on or below the chord connecting any two points:

$$f\{\alpha\mathbf{x}_0 + (1 - \alpha)\mathbf{x}_1\} \leq \alpha f(\mathbf{x}_0) + (1 - \alpha)f(\mathbf{x}_1).$$

■

Theorem 3.1.1 (Existence). Any local minimum of a *convex function* f on a *convex set* $S \subseteq \mathbb{R}^p$ is a global minimum of f on S . □

Theorem 3.1.2 (Uniqueness). A *strictly convex* function f on an unbounded set S has a *unique* global minimum if and only if f is coercive. □

3.2 Optimality Conditions

Definition 3.2.1. For a scalar functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the **gradient** is the column vector:

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$

A point $\mathbf{x}^* \in \mathbb{R}^n$ where $\nabla f(\mathbf{x}^*) = \mathbf{0}$ is a **critical point** of f . ■

Example 3.2.1 (First Order Condition). Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is *continuous differentiable*. For such f , the gradient $\nabla f(\mathbf{x})$ points in the direction of steepest ascent of f . Thus, $-\nabla f(\mathbf{x})$ points in the direction of steepest descent. If \mathbf{x}^* is an *interior* local minimum, than no direction is down hill of f ; thus $\nabla f(\mathbf{x}^*)$ is a critical point of f . This is called the *first order necessary condition* for a minimum. ♠

Definition 3.2.2. For a scalar functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the **Hessian** matrix is:

$$\mathcal{H}(\mathbf{x}) = \frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}'} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}.$$

Notice that in index notation:

$$\left(\frac{\partial^2 f}{\partial \mathbf{x} \partial \mathbf{x}'} \right)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

■

Example 3.2.2 (Second Order Condition). Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable. By Taylor's theorem, there $\exists \alpha \in (0, 1)$ such that:

$$f(\mathbf{x}^* + \mathbf{h}) = f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)\mathbf{h} + \frac{1}{2}\mathbf{h}'\mathcal{H}(\mathbf{x}^* + \alpha\mathbf{h})\mathbf{h}.$$

If \mathbf{x}^* is a critical point, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$:

$$f(\mathbf{x}^* + \mathbf{h}) = f(\mathbf{x}^*) + \frac{1}{2}\mathbf{h}'\mathcal{H}(\mathbf{x}^* + \alpha\mathbf{h})\mathbf{h}.$$

Since f is twice continuously differentiable, for $\alpha \downarrow 0$, the behavior of f near $f(\mathbf{x}^*)$ is determined by \mathcal{H} :

- If $\mathcal{H}(\mathbf{x}^*)$ is positive definite, then $f(\mathbf{x}^*)$ is a local minimum.
- If $\mathcal{H}(\mathbf{x}^*)$ is negative definite, then $f(\mathbf{x}^*)$ is a local maximum.
- If $\mathcal{H}(\mathbf{x}^*)$ is indefinite, then $f(\mathbf{x}^*)$ is a saddle point.

The requirement that $\mathcal{H}(\mathbf{x}^*)$ is positive definite for a local minimum is called the *second order sufficient condition*. ♠

3.2.1 Constrained Optimality Conditions

Definition 3.2.3. A non-zero vector \mathbf{s} is a **feasible direction** at a point $\mathbf{x}^* \in S$ if $\exists \epsilon > 0$ such that $\mathbf{x}^* + \alpha\mathbf{s} \in S$ for $\forall \alpha \in [0, \epsilon]$. ■

Example 3.2.3 (Conditions). The first order necessary condition for \mathbf{x}^* to be a local minimum if f is that:

$$\nabla f(\mathbf{x}^*)'\mathbf{s} \geq 0$$

for any *feasible direction* \mathbf{s} . For an interior point, any direction is feasible, such that the inequality must hold for both \mathbf{s} and $-\mathbf{s}$, recovering $\nabla f(\mathbf{x}^*) = \mathbf{0}$.

The second order necessary condition for \mathbf{x}^* to be a local minimum is that the Hessian $\mathcal{H}(\mathbf{x}^*)$ is positive semi-definite in any feasible direction:

$$\mathbf{s}'\mathcal{H}(\mathbf{x}^*)\mathbf{s} \geq 0.$$

♠

Example 3.2.4 (Lagrange Multipliers). Consider minimizing $f : \mathbb{R}^n \rightarrow \mathbb{R}$ subject to the constraint $g(\mathbf{x}) = 0$, where $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The Lagrange multiplier theorem states that, at the solution \mathbf{x}^* , the gradient $\nabla f(\mathbf{x}^*)$ must lie in the space spanned by the constraint normals:

$$\nabla f(\mathbf{x}^*) = \left\{ \nabla g_1(\mathbf{x}^*), \dots, \nabla g_m(\mathbf{x}^*) \right\} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{pmatrix} = \mathcal{G}(\mathbf{x}^*) \boldsymbol{\lambda}^*,$$

for some coefficients $\boldsymbol{\lambda}^*$. This necessary condition motivates the definition of the Lagrangian $\mathcal{L} : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$:

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

The solution to the original optimization problem satisfies the $n+m$ non-linear equations:

$$\nabla \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \begin{pmatrix} \nabla f(\mathbf{x}^*) + \mathcal{G}(\mathbf{x}^*) \boldsymbol{\lambda}^* \\ g(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}.$$

For a generalization of Lagrange multipliers that allows optimization subject to inequality constraints, see the Karush Kuhn Tucker conditions. ♠

3.3 Unconstrained Optimization Methods

3.3.1 Successive Parabolic Interpolation

Example 3.3.5. Suppose $f : \mathbb{R} \rightarrow \mathbb{R}$ is unimodal on the interval $[x_0, x_2]$. The goal is to locate $x^* = \arg \min_{x \in [x_0, x_2]} f(x)$. Initialize $x_1 \leftarrow (x_0 + x_2)/2$. The Lagrange polynomial passing through these three points is:

$$p(x) = z_0(x - x_1)(x - x_2) + z_1(x - x_0)(x - x_2) + z_2(x - x_0)(x - x_1),$$

where:

$$z_0 = \frac{g(x_0)}{(x_0 - x_1)(x_0 - x_2)}, \quad z_1 = \frac{g(x_1)}{(x_1 - x_0)(x_1 - x_2)}, \quad z_2 = \frac{g(x_2)}{(x_2 - x_0)(x_2 - x_1)}.$$

Setting the derivative of $p(x)$ w.r.t. x to zero gives the update:

$$x_{\text{new}} = \frac{z_0(x_1 + x_2) + z_1(x_0 + x_2) + z_2(x_0 + x_1)}{2(z_0 + z_1 + z_2)}.$$

The remaining logic is to choose 3 of the 4 available points for the next round of interpolation. Consider cases:

1. Suppose $x_0 < x_{\text{new}} < x_1$. If $f(x_{\text{new}}) < f(x_1)$, then x^* lies in $(x_0, x_{\text{new}}, x_1)$. If $f(x_{\text{new}}) > f(x_1)$, then x^* lies in $(x_{\text{new}}, x_1, x_2)$.
2. Suppose $x_1 < x_{\text{new}} < x_2$. If $f(x_{\text{new}}) < f(x_1)$, then x^* lies in $(x_1, x_{\text{new}}, x_2)$. If $f(x_{\text{new}}) > f(x_1)$, then x^* lies in $(x_0, x_1, x_{\text{new}})$.

The iteration continues until $x_2 - x_0 < \epsilon$, or x_1 cases to change. ♠

3.3.2 Gradient Descent

Example 3.3.6. The negative gradient of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ points in the direction of most rapid decrease. Gradient descent chooses the search direction \mathbf{s}_k in the direction of the negative gradient and chooses the step size α_k so as to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$.

Algorithm 8 Gradient Descent

Require: Initialization vector \mathbf{x}_0 .

- 1: **for** $k = 1, \dots$ **do**
 - 2: $\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$.
 - 3: Choose α_k to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$.
 - 4: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$.
 - 5: **end for**
-

♠

3.3.3 Newton Raphson

Example 3.3.7. The second order Taylor approximation to $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at point \mathbf{x} is:

$$f(\mathbf{x} + \mathbf{s}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})' \mathbf{s} + \frac{1}{2} \mathbf{s}' \mathcal{H}(\mathbf{x}) \mathbf{s}.$$

This quadratic form is minimized when:

$$\mathcal{H}(\mathbf{x}) \mathbf{s} = -\nabla f(\mathbf{x}).$$

In principle, Newton's method does not require a step size α_k . However, if initialized far from the minimum, performing a line search for α_k is advisable:

$$\alpha_k = \arg \min_{\alpha} \|f(\mathbf{x}_k + \alpha \mathbf{s}_k)\|_2^2.$$

♠

Algorithm 9 Newton Raphson Iteration**Require:** Initialization vector \mathbf{x}_0 .

- 1: **for** $k = 1, \dots$ **do**
- 2: Solve $\mathcal{H}(\mathbf{x}_k)\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$ for \mathbf{s}_k
- 3: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$.
- 4: **end for**

3.3.4 Quasi-Newton Methods

Example 3.3.8. Quasi-Newton methods avoid the calculation of second derivatives and have the general form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathcal{Q}_k^{-1} \nabla f(\mathbf{x}_k),$$

where \mathcal{Q}_k^{-1} is an approximation to the true Hessian $\mathcal{H}(\mathbf{x}_k)$ at \mathbf{x}_k . The *BFGS* (Broyden, Fletcher, Goldfarb, Shanno) algorithm builds up a successive approximation to the Hessian. Line search along \mathbf{s}_k is optional.

Algorithm 10 BFGS Iteration**Require:** Initialization vector \mathbf{x}_0 .**Require:** \mathbf{B}_0 , often \mathbf{I} .

- 1: **for** $k = 1, \dots$ **do**
- 2: Solve $\mathbf{B}_k \mathbf{s}_k = -\nabla f(\mathbf{x}_k)$ for \mathbf{s}_k .
- 3: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$.
- 4: Set $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.
- 5: Update:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k'}{\mathbf{y}_k' \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k' \mathbf{B}_k}{\mathbf{s}_k' \mathbf{B}_k \mathbf{s}_k}.$$

- 6: **end for**



Example 3.3.9. The *conjugate gradient* algorithm is an important alternative to BFGS which does not require storing an approximation to the Hessian. The search directions are chosen to be orthogonal or conjugate to previous directions, with respect to an inner product that is weighted by accumulated information about the Hessian.

Algorithm 11 Conjugate Gradient

Require: Initialization vector \mathbf{x}_0 .

Require: $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$.

Require: $\mathbf{s}_0 = -\mathbf{g}_0$.

```

1: for  $k = 1, \dots$  do
2:   Choose  $\alpha_k$  to minimize  $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$ .
3:   Update  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$ .
4:   Calculate  $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$ .
5:   Calculate  $\beta_{k+1} = (\mathbf{g}'_{k+1} \mathbf{g}_{k+1}) / (\mathbf{g}'_k \mathbf{g}_k)$ .
6:    $\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{s}_k$ .
7: end for
```



3.4 Nonlinear Least Squares

Example 3.4.10. Given p variables $\boldsymbol{\beta}$ and a collection of n residuals:

$$r_i(\boldsymbol{\beta}) = y_i - f(x_i; \boldsymbol{\beta}),$$

The non-linear least squares estimates of $\boldsymbol{\beta}$ minimize the objective function:

$$Q(\boldsymbol{\beta}) = \frac{1}{2} \mathbf{r}(\boldsymbol{\beta})' \mathbf{r}(\boldsymbol{\beta}).$$

Starting from an initial guess $\boldsymbol{\beta}_0$, the iteration is:

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - (\mathcal{J}'_k \mathcal{J}_k)^{-1} \mathcal{J}'_k \boldsymbol{\beta}_k,$$

where \mathcal{J}_k is the Jacobian matrix evaluated at $\boldsymbol{\beta}_k$:

$$\mathcal{J}_k = \frac{\partial r_i}{\partial \beta_j}(\boldsymbol{\beta}_k).$$

The Gauss-Newton iteration may be written as:

The Levenberg-Marquardt algorithm modifies the Gauss-Newton normal equations in step 2 by introducing a *damping factor* λ :

$$(\mathcal{J}'_k \mathcal{J}_k + \lambda \mathbf{I}) \mathbf{s}_k = -\mathcal{J}'_k \mathbf{r}_k.$$



Algorithm 12 Gauss-Newton**Require:** Initialization vector β_0

```

1: for  $k = 1, \dots$  do
2:   Solve  $\mathcal{J}'_k \mathcal{J}_k \mathbf{s}_k = -\mathcal{J}'_k \mathbf{r}_k$  for  $\mathbf{s}_k$ .
3:    $\beta_{k+1} = \beta_k + \mathbf{s}_k$ .
4: end for

```

3.5 Constrained Optimization**Example 3.5.11.** Recall that the solution to the optimization problem:

$$\arg \min f(\mathbf{x}) \text{ s.t. } \mathbf{g}(\mathbf{x}) = \mathbf{0}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a critical point of the Lagrangian:

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

Thus, the solution \mathbf{x}^* must satisfy:

$$\nabla \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \begin{pmatrix} \nabla f(\mathbf{x}^*) + \mathcal{J}(\mathbf{x}^*) \boldsymbol{\lambda}^* \\ \mathbf{g}(\mathbf{x}^*) \end{pmatrix} = \mathbf{0}$$

where \mathcal{J} is the Jacobian:

$$\mathcal{J}_{ij} = \frac{\partial g_i}{\partial x_j}$$

for some Lagrange multiplier $\boldsymbol{\lambda}^*$.Starting from $(\mathbf{x}_0, \boldsymbol{\lambda}_0)$, on the k th step, the following system is solved for $(\mathbf{s}_k, \boldsymbol{\delta}_k)$:

$$\begin{pmatrix} \mathcal{G}(\mathbf{x}_k, \boldsymbol{\lambda}_k) & \mathcal{J}'_k \\ \mathcal{J}_k & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{s}_k \\ \boldsymbol{\delta}_k \end{pmatrix} = \begin{pmatrix} \nabla f(\mathbf{x}_k) + \mathcal{J}'_k \boldsymbol{\lambda}_k \\ \mathbf{g}(\mathbf{x}_k) \end{pmatrix}.$$

Here:

$$\mathcal{G}(\mathbf{x}, \boldsymbol{\lambda}) = \mathcal{H}_f(\mathbf{x}) + \sum_{j=1}^m \lambda_j \mathcal{H}_{g_j}(\mathbf{x}),$$

where \mathcal{H}_f is the Hessian for f and \mathcal{H}_{g_j} is the Hessian for the j th constraint function g_j .The updates for \mathbf{x}_k and $\boldsymbol{\lambda}_k$ are:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k, \quad \boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \boldsymbol{\delta}_k.$$

An initial value for the Lagrange multipliers may be obtained by solving the $n \times m$ system:

$$\mathcal{J}(\mathbf{x}_0)' \boldsymbol{\lambda} = -\Delta f(\mathbf{x}_0)$$

via ordinary least squares.



Interpolation

4.1 Definition

Definition 4.1.1. Given data $\{(x_i, y_i)\}$, with $x_1 < \dots < x_n$, the goal of **interpolation** is to find a *function* $f: \mathbb{R} \rightarrow \mathbb{R}$ passing through each pair of points:

$$f(x_i) = y_i, \text{ for } i \in \{1, \dots, n\}.$$

■

Discussion 4.1.1. For given data $\{(x_i, y_i)\}$, the interpolating function f is chosen to reside in the linear function space spanned by *basis functions* $\{\phi_1, \dots, \phi_J\}$:

$$f(x) = \sum_{j=1}^J \beta_j \phi_j(x).$$

Requiring that f interpolate the data means $f(x_i) = y_i$ for each $i \in \{1, \dots, n\}$. This requirement translates to a linear system of equations:

$$\mathbf{A}\boldsymbol{\beta} = \mathbf{y}, \tag{4.1.1}$$

where $\mathbf{A} = \mathbf{A}(\mathbf{x})$ is an $n \times J$ *basis matrix*. The existence and uniqueness of a solution $\boldsymbol{\beta}$ to the interpolating equation (4.1.1) depends on the *rank* of \mathbf{A} . ♠

4.2 Monomials

Example 4.2.1. Denote by \mathcal{P}_k the linear space of polynomials of degree at most k , which has dimensions $\dim(\mathcal{P}_k) = k + 1$. The *monomial basis* consists of functions $\phi_j(x) = x^j$, which gives rise to interpolants of the form:

$$p_J(x) = \beta_0 + \beta_1 x + \dots + \beta_J x^J.$$

To interpolate n points $\{(x_i, y_i)\}$ using a polynomial p_{n-1} in \mathcal{P}_{n-1} represented in the monomial basis, the coefficients $\boldsymbol{\beta}$ must satisfy the linear system:

$$\begin{pmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ 2 & x_2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

The matrix \mathbf{A} whose columns are successive powers of a variable \mathbf{x} is called the *Vandermonde* matrix. Such a matrix is non-singular provided all (x_i) are distinct. $\mathbf{A}\boldsymbol{\beta}$ evaluates at n points the polynomial whose coefficients are $\boldsymbol{\beta}$, whereas $\mathbf{A}^{-1}\mathbf{y}$ finds the coefficients of the polynomial whose values at n points are given by \mathbf{y} . Unfortunately, \mathbf{A} becomes *ill-conditioned* or nearly-singular for increasing n . ♠

4.3 Legendre and Newton Polynomials

Definition 4.3.1. For a given set of n data points $\{(x_i, y_i)\}$, the **Lagrange basis** functions for \mathcal{P}_{n-1} , the polynomials of degree at most $n - 1$, are defined by:

$$\phi_i(x) = \frac{\prod_{k \neq i} (x - x_k)}{\prod_{k=1}^n (x_i - x_k)}$$

Observe that $\phi_i(x)$ evaluates to 1 at x_i and 0 at the remaining data points:

$$\phi_i(x_j) = \begin{cases} 0 & j \neq i, \\ 1 & j = i. \end{cases}$$

Thus the basis matrix \mathbf{A} corresponding with the Lagrange basis functions is the identity matrix \mathbf{I} , leading to $\boldsymbol{\beta} = \mathbf{y}$. Overall, the *Lagrange interpolant* is:

$$p_{n-1}(x) = \sum_{i=1}^n y_i \phi_i(x).$$

■

Definition 4.3.2. Given data $\{(x_i, y_i)\}$, the **Newton basis** functions for \mathcal{P}_{n-1} are defined by:

$$\phi_j(x) = \prod_{k=1}^{j-1} (x - x_k)$$

Observe that $\phi_j(x_i) = 0$ for $i < j$, such that the corresponding basis matrix \mathbf{A} is lower triangular. ■

Example 4.3.2 (*Newton Polynomials*). Suppose the data are ordered such that $x_1 < x_2 < \dots < x_n$. The Newton interpolant may be viewed as built up incrementally, starting from $p_1(x) = y_1$, then setting $p_{j+1}(x) = p_j(x) + \beta_{j+1} \phi_{j+1}(x)$. The new coefficient is:

$$\beta_{j+1} = \frac{y_{j+1} - p_j(x_{j+1})}{\phi_{j+1}(x_{j+1})},$$

where $\phi_{j+1}(x_{j+1}) = \prod_{k=1}^j (x_{j+1} - x_k)$. ♠

4.4 Orthogonal Polynomials

Definition 4.4.1. The *inner product* of polynomials p and q on an interval $[a, b]$ is defined by:

$$\langle p, q \rangle = \int_a^b p(t)q(t)w(t)dt,$$

where w is a non-negative weight function. p and q are **orthogonal** if $\langle p, q \rangle = 0$ and **orthonormal** if $\langle p_i, p_j \rangle = \delta_{ij}$, the Kronecker delta. ■

Example 4.4.3. Given a set of polynomials and a weight function w , the polynomials may be orthonormalized using the Gram-Schmidt procedure. Starting from the monomials defined on $[-1, 1]$ with weight function $w(t) = 1$, the Gram-Schmidt procedure yields the *Legendre polynomials*. All orthogonal polynomials satisfy a three-term recurrence:

$$p_{k+1}(t) = (\alpha_k t + \beta_k)p_k(t) - \gamma_k p_{k-1}(t),$$

for some coefficient functions $(\alpha_k, \beta_k, \gamma_k)$. For the Legendre polynomials:

$$(k+1)p_{k+1}(t) = (2k+1)tp_k(t) - kp_{k-1}(t).$$

The *Chebyshev polynomials* (of the first kind) form a basis for polynomials on $[-1, 1]$ using the weight function $w(t) = (1-t^2)^{-1/2}$, and are expressible as:

$$T_k(t) = \cos \{k \arccos(t)\}.$$

The Chebyshev polynomials satisfy the recursion:

$$T_{k+1}(t) = 2 \cdot tT_k(t) - T_{k-1}(t).$$

From the trigonometric definition, the zeros of the Chebyshev polynomials are:

$$t_a = \cos \left\{ \frac{(2a-1)\pi}{2k} \right\}, a \in \{1, \dots, k\},$$

and the extrema are:

$$t_b = \cos \left\{ \frac{b\pi}{k} \right\}, b \in \{0, \dots, k\}$$

These *Chebyshev points* are equally spaced around the rim of the unit circle, yet their projection onto the horizontal axis clusters near the endpoints of the interval $[-1, 1]$. Interpolating a function at the Chebyshev points is preferable to using equispaced interpolation points, as the latter give rise to large oscillations near the endpoints of an interval, known as *Runge's phenomenon*. ♠

4.5 Splines

Definition 4.5.1. Given n data points $\{(x_i, y_i)\}$, **piecewise polynomial** interpolation uses a different polynomial to interpolate each subinterval $[t_i, t_{i+1}]$. The breakpoints (x_i) at which the interpolant changes are called *knots*. ■

Definition 4.5.2. A **spline** is a *piecewise* polynomial of degree k that is $k-1$ times continuously differentiable. ■

Example 4.5.4 (*Natural cubic splines*). A standard cubic polynomial has 4 degrees of freedom. For an unconstrained piecewise polynomial, each knot would introduce an additional 4 degrees of freedom. For the cubic spline, the requirements for continuity of function and its first 2 derivatives impose 3 constraints. Thus for a cubic spline, each additional knot adds 1 degree of freedom, and a cubic spline with n knots has $n + 4$ degrees of freedom. A **natural cubic spline** is additionally constrained to be linear beyond the boundaries of the interpolation interval, removing 2 degrees of freedom for each endpoint. Overall, a natural cubic spline with n knots has n degrees of freedom. ♠

Example 4.5.5 (**B-splines**). As for polynomials, splines may be represented with respect to different bases. The **B-spline** basis uses polynomials that have *local support*, giving rise to a banded system of equations that may be solved stably and efficiently.

For a sequence of knots $\cdots < t_{-1} < t_0 < t_1 < \cdots$, which may extend beyond the range of the observed data, define:

$$B_k^0(t) = \mathbb{I}_{[t_k, t_{k+1})}(t)$$

as the left-closed right-open interval between t_k and t_{k+1} . Define too:

$$\alpha_k^m(t) = \frac{t - t_k}{t_{k+m} - t_k}$$

The B -splines of degree m are defined recursively by:

$$B_k^m(t) = \alpha_k^m(t) \cdot B_k^{m-1}(t) + \{1 - \alpha_{k+1}^m(t)\} \cdot B_{k+1}^{m-1}(t).$$

Taking the integers as the knots $t_k = k$, the degree 0 basis splines are:

$$B_0^0(t) = \mathbb{I}_{[0,1)}(t), \quad B_1^0(t) = \mathbb{I}_{[1,2)}(t), \quad B_2^0(t) = \mathbb{I}_{[2,3)}(t), \quad B_3^0(t) = \mathbb{I}_{[3,4)}(t).$$

The degree 1 auxiliary functions are $\alpha_k^1(t) = (t - k)$. The degree 1 basis splines are:

$$\begin{aligned} B_0^1(t) &= t \cdot \mathbb{I}_{[0,1)}(t) + \{1 - (t - 1)\} \cdot \mathbb{I}_{[1,2)}(t), \\ B_1^1(t) &= (t - 1) \cdot \mathbb{I}_{[1,2)}(t) + \{1 - (t - 2)\} \cdot \mathbb{I}_{[2,3)}(t), \\ B_2^1(t) &= (t - 2) \cdot \mathbb{I}_{[2,3)}(t) + \{1 - (t - 3)\} \cdot \mathbb{I}_{[3,4)}(t). \end{aligned}$$

The degree 2 auxiliary functions are $\alpha_k^2(t) = (t - k)/2$. The degree 2 basis splines are:

$$\begin{aligned} B_0^2(t) &= (t/2) \cdot B_0^1(t) + \{1 - (t - 1)/2\} \cdot B_1^1(t), \\ B_1^2(t) &= (t - 1)/2 \cdot B_1^1(t) + \{1 - (t - 2)/2\} \cdot B_2^1(t). \end{aligned}$$

B-splines have the following properties:

- i. (Local support): For $t < t_k$ or $t > t_{k+m+1}$, $B_k^m(t) = 0$.
- ii. (Positivity): For $t_k < t < t_{k+m+1}$, $B_k^m(t) > 0$.
- iii. (Finite amplitude): $\sum_{k=-\infty}^{\infty} B_k^m(t) = 1$.
- iv. (Smoothness): $B_k^m(t)$ is $k - 1$ times continuously differentiable.
- v. (Basis): The set of functions $\{B_{1-m}^m, \dots, B_{n-1}^m\}$ forms a basis for the set of splines having degree k and knots $t_1 < t_2 < \dots < t_n$.



Numerical Integration and Differentiation

5.1 Quadrature

Definition 5.1.1. Consider the definite integral:

$$I(f) = \int_a^b f(x) dx.$$

An n -point **quadrature rule** is an approximation of the form:

$$\hat{I}(f) = \sum_{i=1}^n \omega_i f(x_i),$$

where $a \leq x_1 < \dots < x_n = b$ are the nodes or quadrature points, and ω_i are the *weights*. ■

Example 5.1.1 (*Interpolatory Integration*). Interpolatory integration rules are obtained by fitting a polynomial of degree $n - 1$ through the n quadrature points $x_1 < \dots < x_n$ then taking the integral of the interpolating polynomial as an approximation to the integral of the original function. The weights ω_i may be chosen by imposing the *moment conditions* that the quadrature rule integrates the first n polynomial basis functions exactly:

$$\begin{aligned} \omega_1 \cdot 1 + \dots + \omega_n \cdot 1 &= \int_a^b 1 dx = b - a, \\ \omega_1 \cdot x_1 + \dots + \omega_n \cdot x_n &= \int_a^b x dx = \frac{1}{2}(b^2 - a^2), \\ &\vdots \\ \omega_1 \cdot x_1^{n-1} + \dots + \omega_n \cdot x_n^{n-1} &= \int_a^b x^{n-1} dx = \frac{1}{n}(b^n - a^n). \end{aligned}$$



5.1.1 Newton-Cotes

Example 5.1.2. **Newton-Cotes** quadrature is characterized by the use of equally-spaced quadrature points. The three simplest quadrature rules are:

- i. The *midpoint rule*, obtained by approximating the function by a degree zero polynomial (i.e. a constant) over the interval, is:

$$\hat{I}_1(f) = (b - a)f\left(\frac{b - a}{2}\right).$$

- ii. The *trapezoid rule*, obtained by approximating the function with a line, is:

$$\hat{I}_2(f) = \frac{(b - a)}{2} \{f(a) + f(b)\}.$$

- iii. *Simpson's rule*, obtained by approximating the function with a quadratic, is:

$$\hat{I}_3(f) = \frac{b - a}{6} \left\{ f(a) + 4f\left(\frac{a + b}{2}\right) + f(b) \right\}.$$

Note that the weights used in Simpson's rule may be obtained by applying the preceding moment conditions to the quadrature points:

$$x_1 = a, \quad x_2 = \frac{a + b}{2}, \quad x_3 = b.$$

Newton-Cotes rules with an odd number of quadrature points n have accuracy 1 degree higher, meaning (e.g.) that the midpoint rule ($n = 1$) can integrate both degree 0 (constant) and degree 1 (linear) polynomials exactly. Simpson's rule ($n = 3$) can integrate degree 2 (quadratic) and degree 3 (cubic) polynomials exactly. However, the trapezoid rule ($n = 2$) can only integrate polynomials up to degree 1 exactly. The improvement in accuracy for odd numbers of quadrature points derives from cancellation of errors on either side of the midpoint of the interval $[a, b]$.



5.1.2 Gaussian

Example 5.1.3. A **Gaussian quadrature** rule is one in which both the quadrature points and their weights are chosen to maximize the degree of polynomial which may be integrated exactly. For example, for a two point rule on $[-1, 1]$, specification of two

nodes x_1, x_2 and two weights ω_1, ω_2 is required. The four moment conditions are:

$$\begin{aligned}\omega_1 + \omega_2 &= \int_{-1}^1 1 dx = 2, \\ \omega_1 x_1 + \omega_2 x_2 &= \int_{-1}^1 x dx = 0, \\ \omega_1 x_1^2 + \omega_2 x_2^2 &= \int_{-1}^1 x^2 dx = \frac{2}{3}, \\ \omega_1 x_1^3 + \omega_2 x_2^3 &= \int_{-1}^1 x^3 dx = 0.\end{aligned}$$

The solution is $x_1 = 1/\sqrt{3}$, $x_2 = -1/\sqrt{3}$, and $\omega_1 = \omega_2 = 1$. Thus:

$$\hat{I}_2(f) = f(-1/\sqrt{3}) + f(1/\sqrt{3}).$$




Discussion 5.1.1. Gaussian quadrature rules are tabulated for standard intervals $[\alpha, \beta]$ such as $[-1, 1]$, and the nodes are typically obtained using orthogonal polynomials. The following change of variables transforms an integral over $[a, b]$ into an integral over $[\alpha, \beta]$:

$$x = \frac{(b-a)u + \alpha\beta - b\alpha}{\beta - \alpha}.$$

Then:

$$\begin{aligned}I(f) &= \int_a^b f(x) dx \\ &= \frac{b-a}{\beta-\alpha} \int_{\alpha}^{\beta} f\left\{\frac{(b-a)u + \alpha\beta - b\alpha}{\beta - \alpha}\right\} du \\ &= \frac{b-a}{\beta-\alpha} \sum_{i=1}^n \omega_i f\left\{\frac{(b-a)u_i + \alpha\beta - b\alpha}{\beta - \alpha}\right\}\end{aligned}$$

where u_i are the quadrature points on $[\alpha, \beta]$ and ω_i are the corresponding weights. 

5.1.3 Piecewise Polynomial

Example 5.1.4. In **piecewise polynomial** quadrature, the interval $[a, b]$ is subdivided into K intervals of length $h = (b-a)/K$, and a simple quadrature rule is applied over each. For example, the piecewise midpoint quadrature rule is:

$$\hat{I}_{1K} = \sum_{k=1}^K (x_k - x_{k-1}) f\left(\frac{x_{k-1} + x_k}{2}\right).$$



5.1.4 Monte Carlo

Discussion 5.1.2. In **Monte Carlo** integration, the function f is sampled at n randomly distributed points on the domain \mathcal{D} , then the integral is approximated by the measure (e.g. area, volume) of the domain $\mu(\mathcal{D})$ scaled by the mean value of the function at the evaluation points:

$$I(f) = \int_{\mathcal{D}} f(\mathbf{x}) d\mathbf{x} \doteq \frac{1}{n} \sum_{i=1}^n f(\mathbf{x}_i) \cdot \mu(\mathcal{D}).$$

For the quadrature rules discussed thus far, the number of required quadrature points grows geometrically with the dimension of the integral. Thus, for a first order accurate method, such as the midpoint rule, increasing the accuracy by a factor of 10 (one decimal point) requires 10^d times more quadrature points, where d is the dimension. Monte Carlo integration has slower, $1/\sqrt{n}$ convergence, meaning that 100 times more points are required to increase the accuracy by one decimal point. However, the rate of convergence of Monte Carlo integration is independent of the dimension, making it a preferred method for high dimensional integrals. ♠

5.2 Differentiation

Example 5.2.5. The **finite difference method** is appropriate for approximating the derivative of a smooth function whose value at sample points may be calculated accurately. For a function $f : \mathbb{R} \rightarrow \mathbb{R}$, consider the following forward and reverse Taylor series expansions:

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{3!}h^3 + \dots, \\ f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{3!}h^3 + \dots. \end{aligned}$$

Notice that the odd powers of h alternate sign. By truncating the series after the first order term, the forward and reverse approximations to $f'(x)$ are:

$$\begin{aligned} f'(x) &\approx \frac{1}{h} \{f(x+h) - f(x)\}, \\ f'(x) &\approx \frac{1}{h} \{f(x) - f(x-h)\}. \end{aligned}$$

Subtracting the two series, which cancels the $f''(x)h^2/2$ component of the residual, gives the *centered difference* approximation:

$$f'(x) \approx \frac{1}{h} \{f(x+h) - f(x-h)\}.$$

An approximation to $f''(x)$ is obtained instead by adding the forward and reverse series, causing the $f'(x)h$ and $f'''(x)h^3/3!$ terms to cancel, then solving for $f''(x)$:

$$f''(x) \approx \frac{1}{h^2} \{f(x+h) + 2f(x) + f(x-h)\}.$$

An alternative derivation is based on the Lagrange interpolation. The Lagrange interpolant of $\{(x_i, y_i), (x_{i+1}, y_{i+1})\}$ is:

$$p(t) = y_i \cdot \frac{t - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \cdot \frac{t - x_i}{x_{i+1} - x_i}$$

Taking the derivative with respect to t gives:

$$\dot{p}(t) = \frac{y_i}{x_i - x_{i+1}} + \frac{y_{i+1}}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{f(x_{i+1}) - f(x_i)}{h},$$

in agreement with the forward difference approximation. This method extends to finding additional and higher order interpolation rules:

- Finding the linear interpolant of $\{(x_{i-1}, y_{i-1}), (x_i, y_i)\}$ then differentiating gives the reverse difference approximation.
- Finding the quadratic interpolant of $\{(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})\}$, differentiating once, then evaluating at $t = x_i$ gives the central difference approximation. Differentiating again gives the approximation to the second derivative.

Beyond simpler generalization to high orders, the Lagrange interpolant approach also allows for unevenly spaced sample points. ♠

Differential Equations

6.1 Ordinary Differential Equations

Definition 6.1.1. For a function $y(t)$ mapping $y : \mathbb{R} \rightarrow \mathbb{R}$, a k th order ordinary differential equation (ODE) takes the general form:

$$F(t, y, \dot{y}, \dots, y^{(k)}) = 0.$$

The ODE is *explicit* if $y^{(k)} = f(t, y, \dot{y}, \dots, y^{(k-1)})$, otherwise *implicit*. ■

Definition 6.1.2. An ODE is **autonomous** if:

$$F(y, \dot{y}, \dots, y^{(k)}) = 0,$$

where F is not a function of the independent variable t . ■

Example 6.1.1. A *explicit* k th order ODE can always be converted into a system of k first order ODEs by defining:

$$u_1(t) = y(t), \quad u_2(t) = \dot{y}(t), \quad \dots, \quad u_k(t) = y^{(k-1)}(t),$$

then:

$$\dot{\mathbf{u}}(t) = \begin{pmatrix} \dot{u}_1(t) \\ \dot{u}_2(t) \\ \vdots \\ \dot{u}_{k-1}(t) \\ \dot{u}_k(t) \end{pmatrix} = \begin{pmatrix} u_2(t) \\ u_3(t) \\ \vdots \\ u_k(t) \\ f(t, u_1, \dots, u_k) \end{pmatrix}. \quad (6.1.2)$$

A non-autonomous ODE can always be made autonomous by defining $u_0 = t$ and appending the identity $\dot{u}_0(t) = 1$ to (6.1.2). ♠

Definition 6.1.3. A system of ODEs is **linear** if $F(t, \mathbf{y}) = \mathbf{A}(t)\mathbf{y} + \mathbf{b}(t)$.

- If \mathbf{A} does not depend on t , the system has *constant coefficients*.
- If $\mathbf{b}(t) = \mathbf{0}$, the system is **homogeneous**.

■

Discussion 6.1.1 (*Existence and Uniqueness*). An initial condition of the form $y(t_0) = y_0$ is necessary but not sufficient for uniqueness of the solution to an ODE. The **Picard-Lindelöf** theorem ensures the existence of a unique solution to the ODE $\dot{y} = f(t, y)$ in an ϵ -neighborhood of t_0 if for every $t \in [t_0 - \epsilon, t_0 + \epsilon]$, $f(t, y)$ is Lipschitz in y with a Lipschitz constant L not depending on t :

$$\|f(t, y_1) - f(t, y_2)\| \leq L\|y_1 - y_2\|.$$

♠

Definition 6.1.4. Consider a potentially non-linear system of ODEs $\dot{\mathbf{y}} = f(t, \mathbf{y})$. The system is **well-posed** if the eigenvalues of the Jacobian $\mathcal{J}_f(t) = \frac{\partial f}{\partial \mathbf{x}'}(t)$ are strictly negative for $\forall t$. The trajectories of a well-posed problem decay towards a stable state as $t \rightarrow \infty$. By contrast, the system is *ill-posed* if the Jacobian encounters positive eigenvalues for at least some values of t . The trajectories of ill-posed problems can grow exponentially far apart as $t \rightarrow \infty$. ■

Example 6.1.2. For a *linear, homogeneous* system of differential equations with *constant coefficients* $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$, satisfying the initial condition $\mathbf{y}(t_0) = \mathbf{y}_0$, a unique solution exists if

\mathbf{A} is diagonalizable. Let \mathbf{v}_i denote the eigenvectors of \mathbf{A} with corresponding eigenvalues λ_i . Let α_i denote coefficients such that:

$$\mathbf{y}_0 = \sum_{i=1}^n \alpha_i \mathbf{v}_i.$$

The solution to the ODE is:

$$\mathbf{y}(t) = \sum_{i=1}^n \alpha_i \mathbf{v}_i e^{\lambda_i t}.$$

Stability is determined by the eigenvalues of \mathbf{A} . Recall that for the 1D problem $\dot{y}(t) = \lambda y(t)$, the solution is $y(t) = y_0 e^{\lambda t}$. The solution diverges to infinity if $\text{Re}(\lambda_i) > 0$, decays to zero if $\text{Re}(\lambda_i) < 0$, and remains bounded if $\text{Re}(\lambda_i) = 0$. If $\text{Im}(\lambda_i) \neq 0$, the solution exhibits oscillations. For the linear system $\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$, the solution is stable if $\text{Re}(\lambda_i) \leq 0$ for all eigenvalues and unstable if any eigenvalue has $\text{Re}(\lambda_i) > 0$. ♠

6.2 Initial Value Problems

Example 6.2.3. The simplest approach to solving ODEs numerically is the **explicit Euler method**:

$$y_{k+1} = y_k + h_k f(t_k, y_k).$$

By contrast, the **implicit Euler method** is:

$$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1}).$$

This method is implicit in that f must be evaluated before y_{k+1} is known. One option is to approximate $y_{k+1}^{(0)}$ by the explicit euler method, then obtain the final estimate y_{k+1} by solving $y - y_k - h_k f(t_{k+1}, y) = 0$ via Newton's method starting from $y = y_{k+1}^{(0)}$.

The advantage of implicit Euler's method is improved stability. Consider the model problem $f(t, y) = \lambda y$ for $\lambda < 0$ and suppose $h_k = h > 0$ is constant. The explicit Euler update is:

$$y_{k+1} = y_k + h\lambda y_k = (1 + h\lambda)y_k = \dots = (1 + h\lambda)^k y_0.$$

Because $\lambda < 0$ by hypothesis, the trajectory y_{k+1} should approach 0 as $k \rightarrow \infty$. However, the explicit Euler iteration only approaches 0 if $0 < |1 + h\lambda| < 1$. In contrast, the implicit euler update is:

$$y_{k+1} = y_k + h\lambda y_{k+1}, \quad (1 - h\lambda)y_{k+1} = y_k, \quad y_{k+1} = \frac{y_0}{(1 - h\lambda)^k}.$$

With $\lambda < 0$, $1 - h\lambda > 0$ thus $y_{k+1} \rightarrow 0$ for any choice of h . The insensitivity of implicit Euler's method to the choice of learning rate h is described as *unconditional stability*. ♠

Example 6.2.4. Higher order update rules can be obtained via Taylor expansion. For example, an explicit 2nd order update rule is:

$$y_{k+1} = y_k + h_k y'_k + \frac{1}{2} h_k^2 y''_k.$$

The second derivative is obtained as the total differential of $y'(t) = f(t, y)$:

$$y'' = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial t} = f_t(t, y) + f_y(t, y) f(t, y).$$

This gives:

$$y'_k = f(t_k, y_k), \quad y''_k = f_t(t_k, y_k) + f_y(t_k, y_k) y'_k.$$



Example 6.2.5. Runge-Kutta methods are higher order updates in which derivatives are approximated by finite differences. For example, *Heun's* method uses:

$$f(t + h, y + h) = f(t, y) + h f_t(t, y) + h f_y(t, y) f(t, y) + \mathcal{O}(h^2),$$

to approximate:

$$y''(t) = f_t(t, y) + h_y(t, y) f(t, y) \approx \frac{1}{h} \{f(t + h, y + h) - f(t, y)\}.$$

The explicit 2nd order update rule is then:

$$\begin{aligned} y_{k+1} &= y_k + h_k y'_k + \frac{1}{2} h_k^2 y''_k \\ &= y_k + h_k y'_k + \frac{1}{2} h_k \{f(t_k + h_k, y_k + h_k) - y'_k\} \\ &= y_k + \frac{1}{2} h_k y'_k + \frac{1}{2} h_k f(t_k + h_k, y_k + h_k). \end{aligned}$$



6.3 Gradient Flow

Discussion 6.3.1. Consider the problem of minimizing the objective function $f(\mathbf{x})$. In *gradient descent*, the objective is minimized starting from some initial point \mathbf{x}_0 by taking discrete steps of size h in the direction of the negative gradient:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - h \nabla f(\mathbf{x}_k).$$

The **gradient flow** $\mathbf{x}(t)$ is the descent trajectory taken by the solution path in the limit as $h \rightarrow 0$, viewed as a continuous function of time. The gradient flow is the solution to the ordinary differential equation:

$$\dot{\mathbf{x}}(t) = -\nabla f\{\mathbf{x}(t)\}.$$

Observe that gradient descent is exactly the forward Euler update applied to the gradient flow ODE.



Discussion 6.3.2. *Moment* methods are developed by breaking the gradient flow ODE into a system of two equations:

$$\dot{\mathbf{x}}(t) = \mathbf{v}(t), \quad \dot{\mathbf{v}}(t) = -\nabla f\{\mathbf{x}(t)\}.$$

Here $\mathbf{v}(t) = \dot{\mathbf{x}}(t)$ represents the velocity, and the negative gradient of the objective function is viewed as a *force* that causes a change $\dot{\mathbf{v}}(t)$ in the *momentum* vector. Note that the change in momentum $\dot{p}(t)$ is identical to the change in velocity $\dot{v}(t)$ under the assumption that mass $m = 1$. ♠

Example 6.3.6. Consider an explicit Euler update for the velocity and an implicit Euler update for the position:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot \mathbf{v}_{k+1}, \quad \mathbf{v}_{k+1} = \mathbf{v}_k - \Delta t \cdot \nabla f\{\mathbf{x}(t)\}.$$

Combining these into a single equation gives:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \cdot \mathbf{v}_k - (\Delta t)^2 \cdot \nabla f\{\mathbf{x}(t)\}.$$

This suggests an update of the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{v}_k - \beta \nabla f\{\mathbf{x}(t)\}.$$

♠

6.4 Boundary Value Problems

Example 6.4.7. Consider the second order ODE:

$$\ddot{y}(t) = f(t, y, \dot{y})$$

with boundary conditions:

$$y(a) = y_0, \quad y(b) = y_1.$$

Introduce a mesh of points $t_k = kh$ between $[0, 1]$, for $k \in \{0, 1, \dots, n\}$ and $h = (b-a)/n$. The *finite difference* approach replaces the derivatives by:

$$\dot{y}(t_k) \approx \frac{y_{k+1} - y_k}{2h}, \quad \ddot{y}(t_k) = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2},$$

and seeks a solution to the system of algebraic equations:

$$\frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} = f\left(t_k, y_k, \frac{y_{k+1} - y_k}{2h}\right)$$

for $k \in \{1, \dots, n-1\}$. The result is an approximation to the solution curve $y(t)$ evaluated at the interval mesh points, i.e. $\{y(t_1), \dots, y(t_{n-1})\}$. ♠

Example 6.4.8. In the *collocation* method, a solution of the following form is sought:

$$\hat{y}(t; \boldsymbol{\beta}) = \sum_{i=1}^n \beta_i \phi_i(t),$$

where $\phi(t)$ are basis functions on $[a, b]$ and the β_i are parameters selected to ensure the differential equation is satisfied at the *collocation points* $a = t_1 < \dots < t_n = b$. Different choices of basis functions are possible: *spectral* methods adopt basis functions with global support, such as trigonometric polynomials, while *finite element* methods adopt basis functions with local support, such as B-splines. The collocation points (t_1, \dots, t_n) need not be regularly spaced. If an exact solution is intractable, an approximate solution may be obtained by defining the residual:

$$r(t, \boldsymbol{\beta}) = \sum_{i=1}^n \beta_i \ddot{\phi}_i(t) - f(t)$$

and selecting β_i to minimize the least squares criterion:

$$Q(\boldsymbol{\beta}) = \frac{1}{2} \int_a^b r^2(t, \boldsymbol{\beta}) dt.$$

By setting the gradient with respect to β_j to zero:

$$\begin{aligned} 0 &= \frac{\partial Q}{\partial \beta_j} = \int_a^b r(t, \boldsymbol{\beta}) \frac{\partial r}{\partial \beta_j} dt \\ &= \int_a^b r(t, \boldsymbol{\beta}) \ddot{\phi}_j(t) dt \\ &= \int_a^b \left\{ \sum_{i=1}^n \beta_i \ddot{\phi}_i(t) - f(t) \right\} \ddot{\phi}_j(t) dt \\ &= \sum_{i=1}^n \left\{ \int_a^b \ddot{\phi}_i(t) \ddot{\phi}_j(t) dt \right\} \beta_i - \int_a^b f(t) \ddot{\phi}_j(t) dt. \end{aligned}$$

This is a symmetric system of linear algebraic equations $\mathbf{A}\boldsymbol{\beta} = \mathbf{b}$,

$$A_{ij} = \int_a^b \ddot{\phi}_i(t) \ddot{\phi}_j(t) dt, \quad b_i = \int_a^b f(t) \ddot{\phi}_i(t) dt,$$

whose solution gives the parameter vector for the approximate solution to the boundary value problem $\hat{y}(t, \boldsymbol{\beta}) = \sum_{i=1}^n \beta_i \phi_i(t)$. ♠

References

- Heath, MT. *Scientific Computing: An Introductory Survey* (2002). <https://epubs.siam.org/doi/book/10.1137/1.9781611975581>.