BROWN MASTER'S IN DATA SCIENCE
DATA 1010 LEARNING OBJECTIVES

Contents

1 Linear Algebra

1.1 Vector spaces

1.1.1 Vectors

1. A **vector** in $\mathbb{R}^n$ is a column of n real numbers. These real numbers are called the **components** of the vector.

2. Vectors can be added together or multiplied by real numbers componentwise.

1.1.2 Linear independence, span, and basis

1. A **linear combination** of a list of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ is an expression of the form

$$c_1\mathbf{v_1} + c_2\mathbf{v_2} + \cdots + c_k\mathbf{v_k},$$

where $c_1, \ldots, c_k$ are real numbers. The c's are called the **weights** of the linear combination.

2. The **span** of a list L of vectors is the set of all vectors which can be written as a linear combination of the vectors in L.

3. A list of vectors is **linearly independent** if and only if the only linear combination which yields the zero vector is the one with all weights zero.

4. A list of two vectors is linearly independent if the vectors are not contained in the same line through the origin, and a list of three vectors is linearly independent if they are not contained in the same plane through the origin.

5. A **vector space** is a nonempty set of vectors which is closed under the vector space operations. Examples in $\mathbb{R}^3$: lines and planes through the origin. If V and W are vector spaces and $V \subset W$, then V is called a **subspace** of W.

6. A list of vectors in a vector space is a **spanning list** if every vector in the vector space can be written as a linear combination of the vectors in that list.

7. A linearly independent spanning list of a vector space is called a **basis** of that vector space.

8. If a vector space has a basis with finitely many vectors in it, then every basis of that space has the same number of vectors. The number of vectors in a basis of a vector space is called the **dimension** of that vector space.

9. If a vector space V has a basis $\mathcal{B} = \{\mathbf{b}_1, \ldots \mathbf{b}_n\}$ and $\mathbf{v} \in V$ is such that

$$\mathbf{v} = v_1\mathbf{b}_1 + \cdots + v_n\mathbf{b}_n,$$

we call $(v_1, \ldots, v_n)$ the coordinates of $\mathbf{v}$ with respect to $\mathcal{B}$.

1.1.3 Linear transformations

1. A **linear transformation** L is a function from one vector space to another which satisfies $L(\alpha\mathbf{v} + \beta\mathbf{w}) = \alpha L(\mathbf{v}) + \beta L(\mathbf{w})$. Geometrically, these are "flat maps": equally spaced lines are mapped to equally spaces lines or points. Examples include scaling, rotation, projection, and reflection.

2 The **rank** of a linear transformation from one vector space to another is the dimension of its range.

3 The **null space** of a linear transformation is the set of vectors which are mapped to the zero vector by the linear transformation.

4 The **rank-nullity theorem** says that the rank of a transformation plus the dimension of its null space is equal to the dimension of its domain.

1.2 Matrix algebra

1 If A is a matrix and $\mathbf{x}$ is a column vector, then $A\mathbf{x}$ is the linear combination of the columns of A with weights given by the entries of $\mathbf{x}$.

2 Linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^m$ are in one-to-one correspondence with $m \times n$ real matrices, where the linear transformation corresponding to A is the one which maps $\mathbf{x} \in \mathbb{R}^n$ to $A\mathbf{x} \in \mathbb{R}^m$.

3 The identity transformation corresponds to the **identity matrix**, which has entries of 1 along the diagonal and zero entries elsewhere.

4 Matrix multiplication corresponds to composition of the corresponding linear transformations: AB is the matrix for which $(AB)(\mathbf{x}) = A(B\mathbf{x})$ for all $\mathbf{x}$.

5 The rank of a matrix is the rank of the corresponding linear transformation, and similarly for the null space.

6 If the columns of a square matrix A are linearly independent, then it has a unique **inverse matrix** A^{-1} with the property that $A\mathbf{x} = \mathbf{b}$ implies $\mathbf{x} = A^{-1}\mathbf{b}$ for all $\mathbf{x}$ and $\mathbf{b}$.

7 Matrix inversion satisfies $(AB)^{-1} = B^{-1}A^{-1}$ if A and B are both invertible.

8 The **transpose** A^T of a matrix A is defined so that the rows of A^T are the columns of A (and vice versa).

9 The transpose is a linear operator: $(cA + B)^\mathsf{T} = cA^\mathsf{T} + B^\mathsf{T}$ if c is a constant and A and B are matrices.

10 The transpose distributes across matrix multiplication but with an order reversal: $(AB)^\mathsf{T} = B^\mathsf{T}A^\mathsf{T}$ if A and B are matrices for which AB is defined.

11 If $A = A^\mathsf{T}$, we say that A is **symmetric**.

1.3 Determinants

1 A linear transformation T from $\mathbb{R}^n$ to $\mathbb{R}^n$ scales all n-dimensional volumes by the same factor: the (absolute value of the) **determinant** of T.

2 The sign of the determinant tells us whether T reverses orientations; e.g., rotations in $\mathbb{R}^2$ have positive determinant, while reflections have negative determinant.

3 $\det AB = \det A \det B$ if A and B are both $n \times n$, because scaling volumes by a factor of k and then by a factor of ℓ is equivalent to scaling volumes by a factor of $k\ell$.

4 $\det A^{-1} = (\det A)^{-1}$, because if A scales volumes by a factor of k, then the inverse of A must scale volumes by a factor of $1/k$.

5 For a square matrix A, the determinant of A is zero if and only if A is noninvertible, because the a linear map from $\mathbb{R}^n$ to $\mathbb{R}^n$ squishes positive-volume regions down to zero-volume regions if and only if it maps $\mathbb{R}^n$ to a lower-dimensional subspace of $\mathbb{R}^n$.

1.4 Dot products and orthogonality

1 The **dot product** of two vectors in $\mathbb{R}^n$ is defined by

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

2 The **length** of a vector $\mathbf{x}$ is defined by $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$.

3 The dot product has a geometric connection with the angle θ between two vectors $\mathbf{x}$ and $\mathbf{y}$, via

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos\theta.$$

4 $\mathbf{x} \cdot \mathbf{y} = 0$ if and only if $\mathbf{x}$ and $\mathbf{y}$ are orthogonal.

5 The dot product is linear: $\mathbf{x} \cdot (c\mathbf{y} + \mathbf{z}) = c\mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}$.

6 The **orthogonal complement** of a vector space V is the set of vectors which are orthogonal to every vector in V.

7 The orthogonal complement of the span of the columns of a matrix A is equal to the null space of A^T.

8 A key application of orthogonality: minimization of $\|A\mathbf{x} - \mathbf{b}\|$ for given A and $\mathbf{b}$.

1. If $\widehat{\mathbf{b}} = A\widehat{\mathbf{x}}$ is the vector in the range of A for which $\|A\mathbf{x} - \mathbf{b}\|$ is minimized, then $\mathbf{b} - \widehat{\mathbf{b}}$ will be orthogonal to the range of A
2. This gives the equation $A^\mathsf{T}(\mathbf{b} - \widehat{\mathbf{b}}) = 0$, which implies $\boxed{\widehat{\mathbf{x}} = (A^\mathsf{T}A)^{-1}A^\mathsf{T}\mathbf{b}}$ if $A^\mathsf{T}A$ is invertible.

9 A square matrix U is said to be **orthogonal** if its columns have unit length and are pairwise orthogonal.

1. Equivalently, U is orthogonal if $U^\mathsf{T}U$ is the identity matrix.
2. Geometrically, an orthogonal matrix U transforms $\mathbb{R}^n$ in a length-preserving and angle-preserving way. You can think of an orthogonal transformation as a rotation/reflection.

10 If U is an $n \times p$ matrix with orthonormal columns, then UU^T is the matrix of the transformation which projects each vector in $\mathbb{R}^n$ onto the p-dimensional subspace of $\mathbb{R}^n$ spanned by the columns of U

1.5 Eigenvalues and matrix diagonalization

1 An eigenvector $\mathbf{v}$ of an $n \times n$ matrix A is a *nonzero* vector with the property that $A\mathbf{v} = \lambda\mathbf{v}$ for some $\lambda \in \mathbb{R}$ (in other words, A maps $\mathbf{v}$ to a vector which is either zero or parallel to $\mathbf{v}$). We call λ an **eigenvalue** of A.

2 If an $n \times n$ matrix A has n linearly independent eigenvectors, we can make a matrix V with these eigenvectors as columns, so that

$$AV = V\Lambda,$$

where Λ is a matrix whose diagonal entries are the eigenvalues (in order corresponding to the columns of V) and whose other entries are zero. We conclude that $A = V\Lambda V^{-1}$, where Λ is a diagonal matrix, and say that A is **diagonalizable**.

3 Diagonalizing a matrix is useful because it identifies axes along which A acts by pure scaling. For example:

1. The inverse of A may be obtained by inverting the diagonal elements of Λ.
2. The square root of A may be obtained by square rooting the diagonal elements of Λ.

4 A **positive definite** matrix A is a symmetric matrix whose eigenvalues are all positive. A **positive semidefinite** matrix A is one whose eigenvalues are all nonnegative. Equivalently, a matrix A is positive semidefinite if $\mathbf{x}^\mathsf{T} A\mathbf{x} \geq 0$ for all $\mathbf{x}$.

5 *Negative definite* and *negative semidefinite* matrices are defined analogously.

6 If A is an $m \times n$ matrix, then $A^\mathsf{T}A$ is its **Gram matrix**. Every matrix of the form $A^\mathsf{T}A$ is positive semidefinite, and the rank of A is equal to the rank of $A^\mathsf{T}A$.

7 If A is a symmetric matrix, then A is *orthogonally* diagonalizable, meaning that

$$A = V\Lambda V^\mathsf{T},$$

where V is an orthogonal matrix.

1.6 Singular value decomposition

1 A square matrix A can be written as a product of two orthogonal matrices and a diagonal matrix with positive diagonal entries.

2 This is because the matrix $\sqrt{A^\mathsf{T}A}$ can be orthogonally diagonalized, and A and $\sqrt{A^\mathsf{T}A}$ are related by an orthogonal transformation.

3 This idea extends to rectangular matrices: for any $m \times n$ matrix A, there exist orthogonal matrices U and V and an $m \times n$ diagonal matrix Σ such that

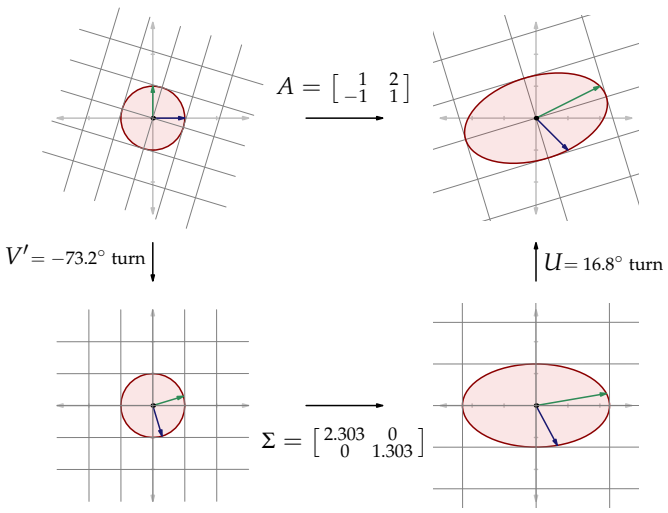
$$A = U\Sigma V^\mathsf{T}. \tag{1.6.3.1}$$

We call (1.6.3.1) a **singular value decomposition** of A. The diagonal entries of Σ are called the **singular values** of A.

4 Geometrically, the singular value decomposition of a matrix expresses the corresponding transformation as a composition of

1. a rotation/reflection,
2. a nonnegative scaling* along the coordinate axes,
3. another rotation/reflection.

The SVD is useful because it enables us to express any transformation as a composition of simple transformations.



* If $m \neq n$, then we must expand the notion of "scaling" to include deletion or insertion of axes

5 The image under a linear transformation $A : \mathbb{R}^n \to \mathbb{R}^m$ of the unit ball in $\mathbb{R}^n$ is an *ellipsoid* in $\mathbb{R}^m$. The singular values of A are the semi-axes of this ellipsoid.

6 **Properties of the SVD**

1. A maps each column of V to the corresponding singular value times the corresponding column of U.
2. The column of V with the largest singular value is the one that stretches the most under A, and the column of V with the least singular value stretches the least.
3. The span of the first k columns of U is the k-dimensional vector space with minimal sum of squared distances to the points represented by the columns of A.

7 The **Moore-Penrose pseudoinverse** A^+ of an $m \times n$ matrix A is defined to be $V\Sigma^+U$, where Σ^+ is the transpose of the matrix obtained by inverting each nonzero element of Σ. The pseudoinverse is a generalization of the inverse:

1. If A is square and invertible, then $A^+ = A^{-1}$
2. If $A\mathbf{x} = \mathbf{b}$ has no solution, then $A^+\mathbf{b}$ is the value of $\mathbf{x}$ which minimizes $\|A\mathbf{x} - \mathbf{b}\|^2$.
3. If $A\mathbf{x} = \mathbf{b}$ has multiple solutions, then $A^+\mathbf{b}$ is the solution with minimal length.

1.7 Matrix Norms

1 The **operator norm** $\|A\|$ of an $m \times n$ matrix A is defined to be the largest value of $\|A\mathbf{v}\|$ for any unit vector $\mathbf{v} \in \mathbb{R}^n$.

2 The operator norm of a matrix is equal to its largest singular value.

2 Multivariable Calculus

2.1 Partial differentiation

1. The partial derivative $\frac{\partial f}{\partial x}(x_0, y_0)$ of a function $f(x, y)$ at a point (x_0, y_0) is the slope of the graph of f in the x-direction at the point (x_0, y_0). In other words, it's the slope of the trace of the graph in the plane $y = y_0$

2. $f_x(x_0, y_0)$ is alternate notation for $\frac{\partial f}{\partial x}(x_0, y_0)$

3. To find the partial derivative with respect to x, we hold y constant and differentiate as usual, and to find the partial derivative with respect to y, we hold x constant and differentiate with respect to y

4. The gradient ∇f of a function f is obtained by putting all the partial derivatives of a function f together into a vector

5. The directional derivative of f in the $\mathbf{u}$ direction is equal to $\nabla f \cdot \mathbf{u}$

6. Because of the formula $D_{\mathbf{u}} f = \nabla f \cdot \mathbf{u} = |\nabla f| \cos\theta$, the direction of the gradient is the direction in which f increases most rapidly. The direction opposite to the gradient is the direction of maximum decrease, and the directions orthogonal to these are the ones in which f is constant

7. The equation of the plane tangent to the graph of a differentiable function f at the point $(a, b, f(a, b))$ is given by

$$z = f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b) = f(a, b) + (\nabla f)(a, b) \cdot (\langle x, y \rangle - \langle a, b \rangle).$$

8. The above equation says how f behaves for values of (x, y) very close to (a, b): the output changes by the x-change $x - a$ times f's sensitivity to changes in x (namely $f_x(a, b)$) *plus* the y-change times f's sensitivity to changes in y (namely $f_y(a, b)$)

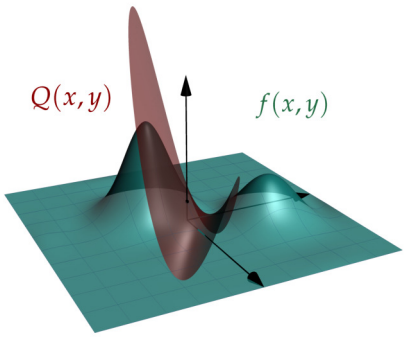
9. The **Hessian** of a function $f : \mathbb{R}^n \to \mathbb{R}$ is the matrix

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

10. The best quadratic approximation to the graph of a twice-differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ at the origin is

$$Q(\mathbf{x}) = f(\mathbf{0}) + (\nabla f(\mathbf{0}))^{\mathsf{T}} \mathbf{x} + \frac{1}{2} \mathbf{x}^{\mathsf{T}} \mathbf{H}(\mathbf{0}) \mathbf{x}.$$

The same is true at points $\mathbf{a}$ other than the origin if we evaluate the gradient and Hessian at $\mathbf{a}$ instead of $\mathbf{0}$ and if we replace $\mathbf{x}$ with $\mathbf{x} - \mathbf{a}$.



2.2 Optimization

1. The **extreme value theorem** guarantees that a continuous function on a closed and bounded region realizes an absolute maximum and an absolute minimum

2. **Critical points** of a function are points in the interior of the function's domain where the gradient of the function is equal to zero, or else where the function is not differentiable

3. If a function has a local minimum or a local maximum at a point, then either that point is a critical point, or it is on the boundary of the domain of that function

4. At a given critical point **a**, a function has

   1. a local maximum if the Hessian at **a** is negative definite
   2. a local minimum if the Hessian at **a** is positive definite
   3. neither a local min nor max if the Hessian at **a** has at least one positive and one negative eigenvalue

5. **Constrained optimization**. Suppose $n \geq 2$ and f is a function defined on the level set of some function $g : \mathbb{R}^n \to \mathbb{R}$.

   1. If f has a local minimum or maximum at a point, then the equation

   $$\nabla f = \lambda \nabla g \tag{2.2.5.1}$$

   is satisfied at that point.

   2. We can use this fact to find the minima and maxima of f: solve (2.2.5.1) together with the given constraint equation $g(x_1, \ldots, x_n) = c$. This is called the **method of Lagrange multipliers**.

   3. The intuition: the equation $\nabla f = \lambda \nabla g$ holds at any maximum because if ∇f were not orthogonal to the level set of g at a given point, we could move along the level set in a direction which has a positive dot product with ∇f. This would increase the function a bit. Same idea for minima.

2.3 Matrix differentiation

1. Suppose $\mathbf{y}$ is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. Writing $\mathbf{y}(\mathbf{x}) = \mathbf{y}(x_1, \ldots, x_n)$, we define the Jacobian matrix to be*

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

* This is called the *numerator layout* convention. Sometimes you'll see the transpose of this matrix instead. See the Wikipedia article on matrix calculus for details.

2. Basic matrix differentiation rules:

$$\frac{\partial}{\partial \mathbf{x}}(A\mathbf{x}) = A$$
$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^\mathsf{T} A) = A^\mathsf{T}$$
$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{u}^\mathsf{T}\mathbf{v}) = \mathbf{u}^\mathsf{T}\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^\mathsf{T}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

3 The Hessian of a function $f : \mathbb{R}^n \to \mathbb{R}$ may be written in terms of matrix differentiation as follows:

$$\mathbf{H}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \left(\frac{\partial f}{\partial \mathbf{x}} \right)^{\mathsf{T}}.$$

2.4 Multivariable integration

1 Suppose f is a function from a region D in $\mathbb{R}^n$ to $\mathbb{R}$. We split the region of integration into many tiny pieces, multiply the volume* of each piece by the value of the function at some point on that piece*, and add up the results. If we take the number of pieces to ∞ and the piece size to zero, then this sum should converge to a number, and if it does then we declare that number to be the value of the **integral** of f, denoted $\int_D f$.

2 Examples:

1. If $f : D \to \mathbb{R}$ represents the **mass** density of a body occupying a region D, then the product of the volume of a small region by the value of f at a point in the small region approximately equals the mass in that region. Therefore, $\int_D f$ represents the *total mass* of the body.
2. If $f : D \to \mathbb{R}$ represents the **probability** density of an $\mathbb{R}^n$-valued random variable, then for any $A \subset D$, restricting f to A and integrating gives the probability that the random variable lies in A.

3 To find the integral of a function f defined on a 2D region D, we set up a double iterated integral over D: the bounds for the outer integral are the smallest and largest values y can be for any point in D, and the bounds for the inner integral are the smallest and largest values x can be for any point in *each "y = constant" slice* of the region.

4 To set up an integral of a function over a 3D region (for the order $\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$): the bounds for the outer integral are the smallest and largest values of z for any point in the region of integration, then the bounds for the middle integral are the smallest and largest values of y for any point in the region in each "z = constant" plane, and the inner bounds are the smallest and largest values of x for any point in the region in each "(y, z) = constant" line.

2.5 Multivariable chain rule and change-of-variables

1 If we compose a function $\mathbf{r} : \mathbb{R}^1 \to \mathbb{R}^2$ with a function $f : \mathbb{R}^2 \to \mathbb{R}^1$, we get a function whose derivative is

$$(f \circ \mathbf{r})'(t) = (\nabla f)(\mathbf{r}(t)) \cdot \mathbf{r}'(t).$$

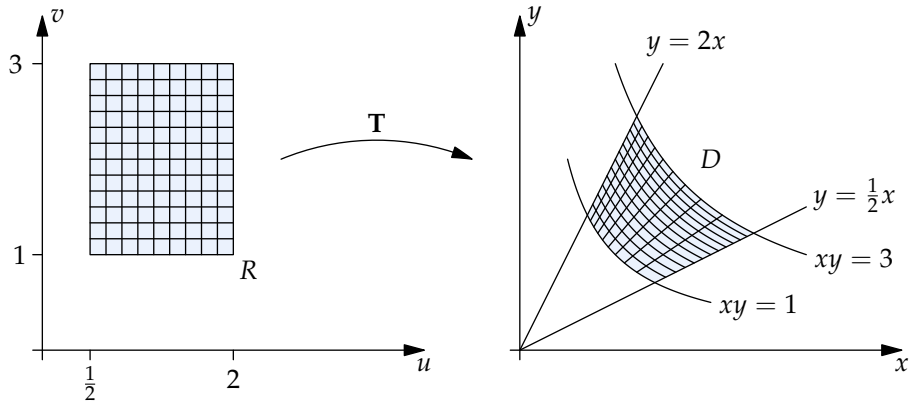
2 The chain rule and directional differentiation and linearization are just different ways of saying the same thing: to find how much f changes when we make a small move from $\mathbf{r}(t)$ to $\mathbf{r}(t) + \mathbf{r}'(t)\Delta t$, we dot the gradient of f with the small step $\mathbf{r}'(t)\Delta t$

3 **The Jacobian determinant of a transformation**.

1. If $\mathbf{T}$ is a transformation from a subset of $\mathbb{R}^n$ to a subset of $\mathbb{R}^n$, then its **Jacobian determinant** measures how it distorts volumes at each point.
2. The Jacobian determinant is the determinant of the Jacobian matrix $\frac{\partial \mathbf{T}(\mathbf{x})}{\partial \mathbf{x}}$.
3. The Jacobian determinant is used when changing regions of integration in higher dimension: *Suppose that* $\mathbf{T} : \mathbb{R}^2 \to \mathbb{R}^2$ *is a differentiable transformation that maps a region R one-to-one onto a region*

D. Then for any continuous function f, we have

$$\iint_D f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \iint_R f(\mathbf{T}^{-1}(x,y))\left|\frac{\partial \mathbf{T}(u,v)}{\partial(u,v)}\right|\,\mathrm{d}u\,\mathrm{d}v.$$



3 Numerical Computation

3.1 Machine arithmetic

1 **Numeric representations**. Computers store all information, including numerical values, as sequences of bits. The **type** of a numeric value specifies how to interpret the underlying sequence of bits as a number.

1. `Int64` (64-bit integers). For $0 \leq n \leq 2^{63} - 1$, we represent n using its binary representation, with leading zeros as necessary to get 64 total bits. For $1 \leq n \leq 2^{63}$, we represent $-n$ using the binary representation of $2^{64} - n$. `Int64` arithmetic wraps around: the `Int64` result of an operation is its integer result shifted by a suitable multiple of 2^{64} to get it in the interval $[-2^{63}, 2^{63})$.

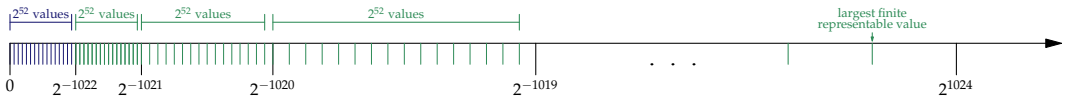
2. `Float64` (64-bit floating point numbers). For each 64-bit sequence, we define σ to be the first bit of the sequence, e to be the next 11 bits interpreted as a binary integer, and f to be the remaining 52 bits interpreted as a binary integer. If $0 < e < 2047$, then the sequence represents the number

$$(-1)^{\sigma} \left(1 + \left(\frac{1}{2}\right)^{52} f\right) \cdot 2^{e-1023}.$$

If $e = 0$, then the sequence represents

$$(-1)^{\sigma} \left(0 + \left(\frac{1}{2}\right)^{52} f\right) \cdot 2^{-1022},$$

If $e = 2047$, then the sequence represents one of the special values `Inf` or `NaN`, depending on the value of f. The nonnegative representable numbers are laid out as follows (figure not drawn to scale!):



3. `Float32` (32-bit floating point numbers). Each 64-bit sequence represents the value

$$(-1)^{\sigma}(1 + f) \cdot 2^{e-127},$$

where σ is the first bit of the sequence, e is the next 8 bits interpreted as a binary integer, and f is the remaining 23 bits are interpreted as a binary fraction.

4. `BigInt/BigFloat` (arbitrary precision numbers). The number of bits in the representation of an arbitrary precision number is not fixed. `BigInt` values are useful for when dealing with vary large integers, and `BigFloat` values are useful when very high precision is required.

2 Choice of numerical representation depends on the application at hand, since each has its advantages and disadvantages.

1. Basic `Float64` operations return the same result as computing the mathematical result of the operation and rounding to the nearest `Float64`-representable value.

2. A *numeric literal* is a sequence of characters to be parsed and interpreted as a numerical value.
   (1) Numeric literals with a decimal point are *real literals*.

(2) Numeric literals without a decimal point are *integer literals*.

Integer literals are interpreted as `Int64` values, and real literals are interpreted as `Float64` values.

3. `Float64` and `Int64` operations are performed *in hardware*, meaning that the use instructions programmed directly into the computer's microprocessor. They are much faster and more memory efficient than arbitrary precision arithmetic, which has to be done in software*:

* `@elapsed` returns the number of seconds an expression takes to evaluate

```
julia> A = [i^2 for i=1:1_000_000]
julia> B = [BigInt(a) for a in A]
julia> @elapsed(sum(B))/@elapsed(sum(A))
25.681853565096898
```

3.2 Error

1 **Error** is the discrepancy between a quantity and the value used to represent it in the program. A result is **accurate** if its error is small. If $\widehat{A}$ is an approximation for A, then

- the **absolute error** is $\widehat{A} - A$, and
- the **relative error** is $\frac{\widehat{A}-A}{A}$.

2 **Sources of numerical error**

1. **Roundoff error**, from rounding numbers to fit them into a floating point representation.
2. **Truncation error**, from using approximate mathematical formulas or algorithms.
3. **Statistical error**, from using randomness in an approximation.

3 **Condition number**

1. The **condition number** of a function measures how it transforms relative errors: it is defined to be the absolute value of the ratio of the relative change in output of the function to a very small* relative change in the input.

* "very small" means that we define the condition number to be the *limit* of the stated ratio as the relative change in input goes to 0

2. The condition number of a *problem* is the condition number of the function which maps the problem's initial data to its solution.
3. If S is the map from the initial data $a \in \mathbb{R}$ to the solution $\mathbf{S}(a) \in \mathbb{R}^n$, then the condition number κ is

$$\kappa(a) = \frac{|a|\|\mathbf{S}'(a)\|}{\|\mathbf{S}(a)\|}. \tag{3.2.3.1}$$

4. Power functions ($a^2, \sqrt{a}, \frac{1}{a}$) have constant condition number, while $a - 1$ has condition number $|\frac{a}{a-1}|$, which is very large near $a = 1$.
5. If the condition number of a problem is very large, then small errors in the problem data lead to large changes in the result. A problem with large condition number is said to be **ill-conditioned**. Unless the initial data can be specified with correspondingly high precision, it will not be possible to solve the problem meaningfully.*

* This would be true even if we could compute with infinite precision arithmetic. Condition number is a property of a *problem*, not of the method used to solve it.

6. The condition number of an $m \times n$ matrix A is defined to be the maximum condition number of the function $\mathbf{x} \mapsto A\mathbf{x}$ as $\mathbf{x}$ ranges over $\mathbb{R}^n$. The condition number of A can be computed as the quotient of its largest and smallest singular values.

7. **Machine epsilon**, denoted ϵ_{mach}, is the maximum relative error associated with rounding a real number to the nearest value representable as a given floating point type. For `Float64`, this value is $\epsilon_{\text{mach}} = 2^{-53} \approx 1.11 \times 10^{-16}$.

8. Since we typically introduce a relative error on the order of ϵ_{mach} to encode the initial data of a problem, the relative error of the computed solution should be expected to be no smaller than $\kappa\epsilon_{\text{mach}}$, regardless of the algorithm used.

9. An algorithm used to solve a problem is **stable** if it is approximately as accurate as the condition number of the problem allows. In other words, an algorithm is *un*stable if the answers it produces have relative error many times larger than $\kappa\epsilon_{\text{mach}}$.

10. Example: calculating the function $f(x) = \sqrt{1+x} - 1$ is well-conditioned near $x = 0$, since the condition number of f converges to 1 as $x \to 0$. The algorithm of substituting directly into f (add 1, square root, subtract 1) includes an ill-conditioned step and is therefore not stable. The algorithm of substituting into the equivalent expression $\frac{x}{\sqrt{1+x}+1}$ is stable, since all of the steps involved in that calculation are well-conditioned.

4 **Hazards**

1. Integer or floating point arithmetic can **overflow**, and may do so without warning.

```
julia> 2^63
-9223372036854775808
julia> 10.0^309
Inf
```

2. Addition of values which are nearly additive opposites can result in a sharp increase in relative error (*catastrophic cancellation*).

3. Relying on exact comparisons for floating point numbers can be dangerous because of roundoff error:

```
julia> a = 1.0
julia> for i = 1:100
         a = a + 0.01
       end
julia> a
2.000000000000001
julia> a > 2
true
```

3.3 Pseudorandom number generation

1 When random numbers are needed in a scientific computing application, we generally use deterministic processes which mimic the behavior of random processes. These are called **pseudo-random number generators** (PRNG).

2 A PRNG takes an initial value, called the **seed**, and uses it to produce a sequence of numbers which are supposed to "look random".

3 If you want your program to be reproducible, then be sure to provide an explicit seed. If you do not, then the seed will be different each time the program runs (typically the seed is based on the current time when unspecified).

4. A simple PRNG is the **linear congruential generator**: fix a large prime number M and an integer c, and consider a seed $a \in \{0, 1, \ldots, M-1\}$. We return the sequence $X_0, X_1, X_2, \ldots$, where $X_0 = a$ and $X_n = \text{mod}(aX_{n-1} + c, m)$, where $\text{mod}(n, d)$ denotes the remainder when n is divided by d.

5. PRNGs are subjected to *statistical tests*, which aim to distinguish the pseudo-random sequence from a random sequence. The idea is to check that features occur with the expected frequency.

3.4 Automatic differentiation

1. Estimating derivatives using difference quotients is numerically unstable.

2. Evaluating derivatives symbolically is computationally infeasible for many of the kinds of functions that arise in modern machine learning applications

3. **Automatic differentiation** is an alternative, which is fast, precise, and scalable. We substitute $\begin{bmatrix} x & 1 \\ 0 & x \end{bmatrix}$ into the function and read off the top-right entry to get the derivative.

4. The downside is that we can't treat f like a black box—all functions that f uses internally must be able to act on these 2×2 matrices, not just numbers.

3.5 Optimization

1. **Gradient descent** is an approach to finding the minimum of a function f from $\mathbb{R}^n$ to $\mathbb{R}$. The basic idea is to repeatedly step in the direction of $-\nabla f$, beginning with some initial guess $\mathbf{x}_0 \in \mathbb{R}^n$.

2. We choose a **learning rate** ϵ and set $\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon \nabla f(\mathbf{x}_{n-1})$. We set a threshold and stop when $\|\nabla f\|$ goes below it.

3. Gradient descent is not guaranteed to find the global minimum since the search can get stuck in a local minimum.