Mathematical and Statistical Foundations (MSF)

Preamble

A strong mathematical foundation remains a bedrock of computer science education and infuses the practice of computing whether in developing algorithms, designing systems, modeling real-world phenomena, or computing with data. This Mathematical and Statistical Foundations knowledge (MSF) area – the successor to the ACM CS 2013 curriculum's "Discrete Structures" area – seeks to identify the mathematical and statistical material that undergirds modern computer science. The change of name corresponds to a realization both that the broader name better describes some of the existing topics from 2013 and that some growing areas of computer science, such as artificial intelligence, machine learning, data science, and quantum computing, have continuous mathematics as their foundations too. Because consideration of additional foundational mathematics beyond traditional discrete structures is a substantial matter, the MSF sub-committee included members who have taught courses in continuous mathematics.

The committee considered the following inputs in order to prepare their recommendations:

- A survey distributed to computer science faculty (nearly 600 respondents) across a variety of institutional types and in various countries;
- Math-related data collected from the of survey of industry professionals (865 respondents);
- Math requirement stated by all the knowledge areas in the report;
- Direct input sought from CS theory community; and
- Review of past reports including recent reports on data science (e.g., Park City report) and quantum computing education.

Changes since CS 2013

The breadth of mathematics needed has grown beyond discrete structures to address the mathematical needs of rapidly growing areas such as artificial intelligence, machine learning, robotics, data science, and quantum computing. These areas call for a renewed focus on probability, statistics, and linear algebra, as supported by the faculty survey that asked respondents to rate various mathematical areas in their importance for both an industry career as well as for graduate school; the combined such ratings for probability, statistics, and linear algebra were 98%, 98% and 89% respectively, reflecting a strong consensus in the CS academic community.

Core Hours

Acknowledging some tensions

Several challenges face CS programs when weighing mathematical requirements: (1) many CS majors otherwise engaged in CS, perhaps aiming for a software career, are unenthusiastic about investing in math; (2) institutions such as liberal-arts colleges often limit how many courses a major may require,

while others may have common engineering courses that fill up the schedule; and (3) some programs adopt a more pre-professional curricular outlook while others emphasize a more foundational one. Thus, we are hesitant to recommend an all-encompassing set of mathematical topics as "every CS degree must require." Instead, we outline two sets of *core* requirements, a minimal "CS-core" set suited to credit-limited majors and a more expansive "KA-core" set to align with technically focused programs. The principle here is that, in light of the additional foundational mathematics needed for AI, data science and quantum computing, programs ought to consider as much as possible from the more expansive KA-core version unless there are sound institutional reasons for alternative requirements.

Knowledge Units	**CS Core**	**KA Core**
Discrete Mathematics	29	11
Probability	11	29
Statistics	10	30
Linear Algebra	5	35
Calculus	0	40
Total	**55**	**200**

Rationale for recommended hours

CS-Core. While some discrete math courses include probability, we highlight its importance so that a minimum number of hours (11) is devoted to probability given its increased importance for AI and data science, and the strong consensus in the academic community based on the survey. Taken together, the total CS core across discrete math and probability (40 hours) is typical of a one-semester 3-credit course. 15 hours are allotted to statistics and linear algebra for basic definitions so that, for example, students should at least be familiar with terms like *mean*, *standard deviation* and *vector*. These could be covered in CS courses. Many programs typically include a broader statistics requirement.

KA-Core. Note that the calculus hours roughly correspond to the typical Calculus-I course now standard across the world. Based on our survey, most programs already require Calculus-I. However, we have left out Calculus-II (an additional 40 hours) and leave it to programs to decide whether Calculus-II should be added to program requirements. Programs could choose to require a more rigorous calculus-based probability or statistics sequence, or non-calculus requiring versions. Similarly, linear algebra can be taught as an applied course without a calculus prerequisite or as a more advanced course.

Knowledge Units

MSF-Discrete: Discrete Mathematics

CS Core:

1. Sets, relations, functions, cardinality
2. Recursive mathematical definitions
3. Proof techniques (induction, proof by contradiction)
4. Permutations, combinations, counting, pigeonhole principle
5. Modular arithmetic
6. Logic: truth tables, connectives (operators), inference rules, formulas, normal forms, simple predicate logic
7. Graphs: basic definitions

Illustrative Learning Outcomes:

1. Sets, Relations, and Functions, Cardinality
    - *a.* Explain with examples the basic terminology of functions, relations, and sets.
    - *b.* Perform the operations associated with sets, functions, and relations.
    - *c.* Relate practical examples to the appropriate set, function, or relation model, and interpret the associated operations and terminology in context.
    - *d.* Calculate the size of a finite set, including making use of the sum and product rules and inclusion-exclusion principle.
    - *e.* Explain the difference between finite, countable, and uncountable sets.
2. Recursive mathematical definitions
    - *a.* Apply recursive definitions of sequences or structures (e.g., Fibonacci numbers, linked lists, parse trees, fractals).
    - *b.* Formulate inductive proofs of statements about recursive definitions.
    - *c.* Solve a variety of basic recurrence relations.
    - *d.* Analyze a problem to determine underlying recurrence relations.
    - *e.* Given a recursive/iterative code snippet, describe its underlying recurrence relation, hypothesize a closed form for the recurrence relation, and prove the hypothesis correct (probably using induction).
3. Proof Techniques
    - *a.* Identify the proof technique used in a given proof.
    - *b.* Outline the basic structure of each proof technique (direct proof, proof by contradiction, and induction) described in this unit.
    - *c.* Apply each of the proof techniques (direct proof, proof by contradiction, and induction) correctly in the construction of a sound argument.
    - *d.* Determine which type of proof is best for a given problem.
    - *e.* Explain the parallels between ideas of mathematical and/or structural induction to recursion and recursively defined structures.
    - *f.* Explain the relationship between weak and strong induction and give examples of the appropriate use of each.
4. Permutations, combinations, counting
    - *a.* Apply counting arguments, including sum and product rules, inclusion-exclusion principle and arithmetic/geometric progressions.
    - *b.* Apply the pigeonhole principle in the context of a formal proof.

   c. Compute permutations and combinations of a set, and interpret the meaning in the context of the particular application.
   d. Map real-world applications to appropriate counting formalisms, such as determining the number of ways to arrange people around a table, subject to constraints on the seating arrangement, or the number of ways to determine certain hands in cards (e.g., a full house).
5. Modular arithmetic
   a. Perform computations involving modular arithmetic.
   b. Explain the notion of greatest common divisor, and apply Euclid's algorithm to compute it.
6. Logic
   a. Convert logical statements from informal language to propositional and predicate logic expressions.
   b. Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae, computing normal forms, or negating a logical statement.
   c. Use the rules of inference to construct proofs in propositional and predicate logic.
   d. Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g., program correctness), database queries, and algorithms.
   e. Apply formal logic proofs and/or informal, but rigorous, logical reasoning to real problems, such as predicting the behavior of software or solving problems such as puzzles.
   f. Describe the strengths and limitations of propositional and predicate logic.
   g. Explain what it means for a proof in propositional (or predicate) logic to be valid.
7. Graphs
   a. Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of types of graphs, including trees.
   b. Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees, along with breadth-first and depth-first search for graphs.
   c. Model a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology, the organization of a hierarchical file system, or a social network.
   d. Show how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting.

MSF-Probability: Probability

CS Core:

1. Basic notions: sample spaces, events, probability, conditional probability, Bayes' rule
2. Discrete random variables and distributions
3. Continuous random variables and distributions
4. Expectation, variance, law of large numbers, central limit theorem
5. Conditional distributions and expectation
6. Applications to computing

KA Core:

The recommended topics are the same between CS core and KA-core, but with far more hours, the KA-core can cover these topics in depth and might include more computing-related applications.

Illustrative Learning Outcomes:

1. Basic notions: sample spaces, events, probability, conditional probability, Bayes' rule
    a. Translate a prose description of a probabilistic process into a formal setting of sample spaces, outcome probabilities, and events.
    b. Calculate the probability of simple events.
    c. Determine whether two events are independent.
    d. Compute conditional probabilities, including through applying (and explaining) Bayes' Rule.
2. Discrete random variables and distributions
    a. Define the concept of a random variable and indicator random variable.
    b. Determine whether two random variables are independent.
    c. Identify common discrete distributions (e.g., uniform, Bernoulli, binomial, geometric).
3. Continuous random variables and distributions
    a. Identify common continuous distributions (e.g., uniform, normal, exponential).
    b. Calculate probabilities using cumulative density functions.
4. Expectation, variance, law of large numbers, central limit theorem
    a. Define the concept of expectation and variance of a random variable.
    b. Compute the expected value and variance of simple or common discrete/continuous random variables.
    c. Explain the relevance of the law of large numbers and central limit theorem to probability calculations.
5. Conditional distributions and expectation
    a. Explain the distinction between a joint distribution and a conditional distribution.
    b. Compute conditional distributions from a full distribution, for both discrete and continuous random variables.
    c. Compute conditional expectations for both discrete and continuous random variables.
6. Applications to computing
    a. Describe how probability can be used to model real-life situations or applications, such as predictive text, hash tables, and quantum computation.
    b. Apply probabilistic processes in solving computational problems, such as through randomized algorithms or in security contexts.

MSF-Statistics: Statistics

CS Core:

1. Basic definitions and concepts: populations, samples, measures of central tendency, variance
2. Univariate data: point estimation, confidence intervals

KA Core:

3. Multivariate data: estimation, correlation, regression
4. Data transformation: dimension reduction, smoothing
5. Statistical models and algorithms

Illustrative Learning Outcomes:

CS Core:

1. Basic definitions and concepts: populations, samples, measures of central tendency, variance
    a. Create and interpret frequency tables
    b. Display data graphically and interpret graphs (e.g. histograms)
    c. Recognize, describe and calculate means, medians, quantiles (location of data)
    d. Recognize, describe and calculate variances (spread of data)
2. Univariate data: point estimation, confidence intervals
    a. Formulate maximum likelihood estimation (in linear-Gaussian settings) as a least-squares problem
    b. Calculate maximum likelihood estimates
    c. Calculate maximum a posteriori estimates and make a connection with regularized least squares
    d. Compute confidence intervals as a measure of uncertainty

KA Core:

3. Multivariate data: estimation, correlation, regression
    a. Formulate the multivariate maximum likelihood estimation problem as a least-squares problem
    b. Interpret the geometric properties of maximum likelihood estimates
    c. Derive and calculate the maximum likelihood solution for linear regression
    d. Derive and calculate the maximum a posteriori estimate for linear regression
    e. Implement both maximum likelihood and maximum a posteriori estimates in the context of a polynomial regression problem
    f. Formulate and understand the concept of data correlation (e.g., in 2D)
4. Data transformation: dimension reduction, smoothing
    a. Formulate and derive PCA as a least-squares problem
    b. Geometrically interpret PCA (when solved as a least-squares problem)
    c. Understand when PCA works well (one can relate back to correlated data)
    d. Geometrically interpret the linear regression solution (maximum likelihood)
5. Statistical models and algorithms
    a. Apply PCA to dimensionality reduction problems
    b. Understand the trade-off between compression and reconstruction power
    c. Apply linear regression to curve-fitting problems
    d. Understand the concept of overfitting
    e. Discuss and apply cross-validation in the context of overfitting and model selection (e.g., degree of polynomials in a regression context)

MSF-Linear: Linear Algebra

CS Core:

1. Vectors: definitions, vector operations, geometric interpretation, angles

KA Core:

2. Matrices, matrix-vector equation, geometric interpretation, geometric transformations with matrices
3. Solving equations, row-reduction

4. Linear independence, span, basis
5. Orthogonality, projection, least-squares, orthogonal bases
6. Linear combinations of polynomials, Bezier curves
7. Eigenvectors and eigenvalues
8. Applications to computer science: PCA, SVD, page-rank, graphics

Illustrative Learning Outcomes:

CS Core:

1. Vectors: definitions, vector operations, geometric interpretation, angles
    a. Understand algebraic and geometric representations of vectors in R^n and their operations, including addition, scalar multiplication and dot product
    b. List properties of vectors in R^n
    c. Compute angles between vectors in R^n

KA Core:

2. Matrices, matrix-vector equation, geometric interpretation, geometric transformations with matrices
    a. Perform common matrix operations, such as addition, scalar multiplication, multiplication, and transposition
    b. Relate a matrix to a homogeneous system of linear equations
    c. Recognize when two matrices can be multiplied
    d. Relate various matrix transformations to geometric illustrations
3. Solving equations, row-reduction
    a. Formulate, solve, apply, and interpret properties of linear systems
    b. Perform row operations on a matrix
    c. Relate an augmented matrix to a system of linear equations
    d. Solve linear systems of equations using the language of matrices
    e. Translate word problems into linear equations
    f. Perform Gaussian elimination
4. Linear independence, span, basis
    a. Define subspace of a vector space
    b. List examples of subspaces of a vector space
    c. Recognize and use basic properties of subspaces and vector spaces
    d. Determine whether or not particular subsets of a vector space are subspaces
    e. Discuss the existence of a basis of an abstract vector space
    f. Describe coordinates of a vector relative to a given basis
    g. Determine a basis and the dimension of a finite-dimensional space
    h. Discuss spanning sets for vectors in R^n
    i. Discuss linear independence for vectors in R^n
    j. Define the dimension of a vector space
5. Orthogonality, projection, least-squares, orthogonal bases
    a. Explain the Gram-Schmidt orthogonalization process
    b. Define orthogonal projections
    c. Define orthogonal complements

    d. Compute the orthogonal projection of a vector onto a subspace, given a basis for the subspace
    e. Explain how orthogonal projections relate to least square approximations
6. Linear combinations of polynomials, Bezier curves
    a. Identify polynomials as generalized vectors
    b. Explain linear combinations of basic polynomials
    c. Understand orthogonality for polynomials
    d. Distinguish between basic polynomials and Bernstein polynomials
    e. Apply Bernstein polynomials to Bezier curves
7. Eigenvectors and eigenvalues
    a. Find the eigenvalues and eigenvectors of a matrix
    b. Define eigenvalues and eigenvectors geometrically
    c. Use characteristic polynomials to compute eigenvalues and eigenvectors
    d. Use eigenspaces of matrices, when possible, to diagonalize a matrix
    e. Perform diagonalization of matrices
    f. Explain the significance of eigenvectors and eigenvalues
    g. Find the characteristic polynomial of a matrix
    h. Use eigenvectors to represent a linear transformation with respect to a particularly nice basis
8. Applications to computer science: PCA, SVD, page-rank, graphics
    a. Explain the geometric properties of PCA
    b. Relate PCA to dimensionality reduction
    c. Relate PCA to solving least-squares problems
    d. Relate PCA to solving eigenvector problems
    e. Apply PCA to reducing the dimensionality of a high-dimensional dataset (e.g., images)
    f. Explain the page-rank algorithm and understand how it relates to eigenvector problems
    g. Explain the geometric differences between SVD and PCA
    h. Apply SVD to a concrete example (e.g., movie rankings)

MSF-Calculus

KA Core:

1. Sequences, series, limits
2. Single-variable derivatives: definition, computation rules (chain rule etc), derivatives of important functions, applications
3. Single-variable integration: definition, computation rules, integrals of important functions, fundamental theorem of calculus, definite vs indefinite, applications (including in probability)
4. Parametric and polar representations
5. Taylor series
6. Multivariate calculus: partial derivatives, gradient, chain-rule, vector valued functions, applications to optimization, convexity, global vs local minima
7. ODEs: definition, Euler method, applications to simulation

Note: the calculus topics listed above are aligned with computer science goals rather than with traditional calculus courses. For example, multivariate calculus is often a course by itself but computer science undergraduates only need parts of it for machine learning.

Illustrative Learning Outcomes:

1. Sequences, series, limits
    a. Explain the difference between infinite sets and sequences
    b. Explain the formal definition of a limit
    c. Derive the limit for examples of sequences and series
    d. Explain convergence and divergence
    e. Apply L'Hospital's rule and other approaches to resolving limits
2. Single-variable derivatives: definition, computation rules (chain rule etc), derivatives of important functions, applications
    a. Explain a derivative in terms of limits
    b. Explain derivatives as functions
    c. Perform elementary derivative calculations from limits
    d. Apply sum, product and quotient rules
    e. Work through examples with important functions
3. Single-variable integration: definition, computation rules, integrals of important functions, fundamental theorem of calculus, definite vs indefinite, applications (including in probability)
    a. Explain the definitions of definite and indefinite integrals
    b. Apply integration rules to examples with important functions
    c. Explore the use of the fundamental theorem of calculus
    d. Apply integration to problems
4. Parametric and polar representations
    a. Apply parametric representations of important curves
    b. Apply polar representations
5. Taylor series
    a. Derive Taylor series for some important functions
    b. Apply the Taylor series to approximations
6. Multivariate calculus: partial derivatives, gradient, chain-rule, vector valued functions, applications to optimization, convexity, global vs local minima
    a. Compute partial derivatives and gradients
    b. Work through examples with vector-valued functions with gradient notation
    c. Explain applications to optimization
7. ODEs: definition, Euler method, applications to simulation
    a. Apply the Euler method to integration
    b. Apply the Euler method to a single-variable differential equation
    c. Apply the Euler method to multiple variables in an ODE

Professional Dispositions

We focus on dispositions helpful to students learning mathematics as well as professionals who need to refresh previously learned mathematics or learn new topics.

- ***Growth mindset***. Perhaps the most important of the dispositions, students should be persuaded that anyone can learn mathematics and that success is not based dependent on innate ability.
- ***Practice mindset***. Students should be educated about the nature of "doing" mathematics and learning through practice with problems as opposed to merely listening or observing demonstrations in the classroom.
- ***Delayed gratification***. Most students are likely to learn at least some mathematics from mathematics departments unfamiliar with computing applications; computing departments should acclimate the students to the notion of waiting to see computing applications. Many of the new growth areas such as AI or quantum computing can serve as motivation.
- ***Persistence***. Student perceptions are often driven by frustration with unable to solve hard problems that they see some peers tackle seemingly effortlessly; computing departments should help promote the notion that eventual success through persistence is what matters.

Math Requirements

The intent of this section is to list the likely most important topics that should expected from students entering a computing program, typically corresponding to pre-calculus in high school. We recommend pre-calculus as a prerequisite for discrete mathematics.

Required:

- Algebra and numeracy:
    - Numeracy: numbers, operations, types of numbers, fluency with arithmetic, exponent notation, rough orders of magnitude, fractions and decimals.
    - Algebra: rules of exponents, solving linear or quadratic equations with one or two variables, factoring, algebraic manipulation of expressions with multiple variables.
- Precalculus:
    - Coordinate geometry: distances between points, areas of common shapes
    - Functions: function notation, drawing and interpreting graphs of functions
    - Exponentials and logarithms: a general familiarity with the functions and their graphs
    - Trigonometry: familiarity with basic trigonometric functions and the unit circle

Course Packaging Suggestions

Every department faces constraints in delivering content, which precludes merely requiring a long list of courses covering every single desired topic. These constraints include content-area ownership, faculty size, student preparation, and limits on the number of departmental courses a curriculum can require. We list below some options for mathematical foundations, combinations of which might best fit any particular institution:

- ***Traditional course offerings***. With this approach, a computer science department can require students to take math-department courses in any of the five broad mathematical areas listed above.
- ***A "Continuous Structures" analog of Discrete Structures***. Many computer science departments now offer courses that prepare students mathematically for AI and machine learning. Such courses can combine just enough calculus, optimization, linear algebra and probability; yet others may split

linear algebra into its own course. These courses have the advantage of motivating students with computing applications, and including programming as pedagogy for mathematical concepts.

- ***Integration into application courses***. An application course, such as machine learning, can be spread across two courses, with the course sequence including the needed mathematical preparation taught just-in-time, or a single machine learning course can balance preparatory material with new topics. This may have the advantage of mitigating turf issues and helping students see applications immediately after encountering math.
- ***Specific course adaptations***. For nearly a century, physics and engineering needs have driven the structure of calculus, linear algebra, and probability. Computer science departments can collaborate with their colleagues in math departments to restructure math-offered sections in these areas that are driven by computer science applications. For example, calculus could be reorganized to fit the needs of computing programs into two calculus courses, leaving a later third calculus course for engineering and physics students.

Committee

Chair: Rahul Simha, The George Washington University, Washington DC, USA

Members:

- Richard Blumenthal, Regis University, Denver, CO, USA
- Marc Deisenroth, University College London, London, UK
- Michael Goldweber, Denison University, Granville, OH, USA
- David Liben-Nowell, Carleton College, Northfield, MN, USA
- Jodi Tims, Northeastern University, Boston, MA, USA