Chapter 10: VOID-Constrained Numerical Methods—Axiomatic Framework with Practical Applications

Konrad Wojnowski, PhD
ChatGPT
Claude.ai

September 21, 2024

1 Introduction

VOID-constrained numerical methods introduce the concept of a universal threshold δ_{VOID}, below which computational distinctions become increasingly probabilistic and indistinct. This is vital for optimizing calculations by limiting precision to practical levels, especially in multi-scale problems and hardware-constrained systems. By acknowledging that infinite precision is unattainable and unnecessary due to inherent uncertainties at small scales, these methods integrate the principles of the VOID Granularity Framework (VGF) and Probabilistic Geometry (PG) into traditional numerical techniques such as differentiation and integration.

This chapter establishes a structured framework for VOID-constrained numerical methods, enhanced to reflect the probabilistic nature of computations at scales approaching δ_{VOID}. The axioms presented extend the previous work and provide practical applications for computational fields.

2 Axiom 9.1: VOID-Constrained Numerical Differentiation

In numerical differentiation, the step size h is constrained by δ_{VOID}, ensuring that calculations do not exceed the precision limits imposed by the VOID Granularity Framework.

2.1 Traditional Finite Difference Method (FDM)

The traditional forward finite difference approximation of the derivative is:

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

2.2 VOID-Constrained Adjustment

Under VOID constraints, the step size h is adjusted to account for probabilistic uncertainties at small scales:

$$h = \max(\delta_{\text{VOID}}, h_{\text{optimal}})$$

Where:

- h_{optimal} is the step size determined by error analysis in classical methods.
- $h \geq \delta_{\text{VOID}}$ ensures that h does not fall below the granularity threshold.

The VOID-constrained derivative is then approximated as:

$$f'_{\text{VOID}}(x) \approx \frac{f(x+h) - f(x)}{h}$$

2.3 Probabilistic Consideration

At scales near δ_{VOID}, the differences $f(x+h) - f(x)$ become probabilistic due to inherent uncertainties:

$$f'_{\text{VOID}}(x) = \mathbb{E}\left[\frac{f(x+h) - f(x)}{h}\right]$$

Where $\mathbb{E}$ denotes the expected value.

2.4 Practical Application

This method is highly relevant in multi-scale simulations where excessive refinement leads to probabilistic indistinguishability. For example, in quantum mechanics or cosmological simulations, where precision is limited by fundamental uncertainties, the VOID-constrained FDM ensures that calculations respect physical precision constraints.

3 Axiom 9.2: VOID-Constrained Numerical Integration

Numerical integration in a VOID-constrained system uses a step size h related to δ_{VOID}. The classical trapezoidal rule is modified to respect VOID Granularity and probabilistic uncertainties.

3.1 VOID-Constrained Trapezoidal Rule

$$\int_a^b f(x)\,dx_{\text{VOID}} \approx \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right]$$

Where:

- $h = \max(\delta_{\text{VOID}}, h_{\text{optimal}})$
- $x_i = a + ih$

3.2 Probabilistic Integration

At scales near δ_{VOID}, the function values $f(x_i)$ are subject to probabilistic variations:

$$\int_a^b f(x)\,dx_{\text{VOID}} = \mathbb{E}\left[\sum_{i=0}^{n-1} f(x_i)\cdot h\right]$$

3.3 Practical Application

This approach is especially useful in simulations such as gravitational wave modeling or fluid dynamics, where computations across multiple scales require varying precision, and ultra-fine steps could introduce probabilistic errors without meaningful gains in accuracy.

4 Axiom 9.3: Adaptive VOID Integration

In adaptive integration, the step size h dynamically adjusts according to the complexity of the function being integrated and the VOID Granularity threshold.

4.1 Adaptive Step Size Definition

$$h_{\text{adaptive}}(x) = \begin{cases} \delta_{\text{VOID}}, & \text{if function complexity is high at } x \\ \min(h_{\text{optimal}}(x), H), & \text{if function is smooth at } x \end{cases}$$

Where:

- $h_{\text{optimal}}(x)$ is determined by error estimates.
- H is a maximum allowable step size.
- Ensures $h_{\text{adaptive}}(x) \geq \delta_{\text{VOID}}$.

4.2 Probabilistic Consideration

When $h_{\text{adaptive}}(x) = \delta_{\text{VOID}}$, computations involve probabilistic uncertainties:

$$\int_a^b f(x)\, dx_{\text{VOID}} = \mathbb{E}\left[\sum_i f(x_i) \cdot h_{\text{adaptive}}(x_i)\right]$$

4.3 Practical Application

In quantum field theory simulations or meteorological modeling, where system behaviors vary drastically, adaptive VOID integration optimizes computational resources, focusing precision where it matters most while acknowledging probabilistic limitations.

5 Axiom 9.4: Hardware-Constrained VOID Threshold

The VOID threshold δ_{VOID} is influenced by hardware limitations such as floating-point accuracy and sensor resolution, as well as probabilistic uncertainties inherent in measurements.

5.1 Definition

$$\delta_{\text{system}} = \max(\delta_{\text{hardware}}, \delta_{\text{fundamental}}, \delta_{\text{probabilistic}})$$

Where:

- δ_{hardware} represents precision limitations of the hardware (e.g., bit precision).
- $\delta_{\text{fundamental}}$ represents theoretical limits, such as the Planck length.
- $\delta_{\text{probabilistic}}$ accounts for uncertainties in measurements or computations.

5.2 Practical Application

In fields such as machine learning, high-energy physics, and AI sensory models, where hardware influences precision and measurements are probabilistic, these methods allow computations to respect both hardware limitations and inherent uncertainties, avoiding unnecessary precision where distinctions are meaningless.

6 Axiom 9.5: Adaptive Methods for Multi-Scale Problems

VOID-constrained methods adapt to varying scales within a simulation by adjusting granularity and acknowledging probabilistic uncertainties.

6.1 Adaptive Granularity Definition

$$\delta_{\text{adaptive}}(x) = \begin{cases} \delta_{\text{VOID}}, & \text{for fine-scale behavior} \\ \delta_{\text{coarse}}(x), & \text{for larger-scale behavior} \end{cases}$$

Where:

- $\delta_{\text{coarse}}(x)$ is a larger scale threshold determined by the problem's requirements.
- Ensures computational resources are optimally used where fine precision is unnecessary and acknowledges probabilistic effects at small scales.

6.2 Practical Application

This is particularly useful in simulations such as weather systems or biological modeling, where behavior spans multiple scales, and small-scale phenomena are subject to probabilistic uncertainties. Adapting precision based on scale saves computational power while maintaining accuracy in key areas.

7 Axiom 9.6: Parallelization and Hardware-Aware VOID Algorithms

To optimize VOID-constrained numerical methods for hardware, algorithms must account for both precision constraints and the capabilities of modern computing architectures while considering probabilistic uncertainties at small scales.

7.1 Parallel Processing Adjustment

- Algorithm Optimization: Algorithms are designed to scale with hardware capabilities (e.g., multi-core processors, GPUs) while respecting δ_{VOID}.
- Probabilistic Load Balancing: Workloads are distributed considering the probabilistic nature of computations at different scales.

7.2 Practical Application

In high-performance computing and large-scale simulations (e.g., climate modeling or astrophysics), parallel computing allows for efficient, scalable solutions while managing granularity and probabilistic uncertainties at each level.

8 Axiom 9.7: Parallelization and Hardware-Aware VOID Algorithms

To optimize VOID-constrained numerical methods for hardware, algorithms must account for precision constraints and leverage the capabilities of modern computing architectures while managing probabilistic uncertainties at small scales.

8.1 Mathematical Framework

8.1.1 Algorithm Optimization

Objective: Efficiently utilize hardware resources (e.g., multi-core processors, GPUs) while ensuring step sizes h adhere to the VOID Granularity threshold δ_{VOID}.

Definitions:

- $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$: Set of computational tasks.
- $\mathcal{H} = \{H_1, H_2, \ldots, H_n\}$: Set of hardware units (cores/GPUs).
- C_j: Computational capacity of hardware unit H_j.
- λ_i: Computational load of task T_i.
- U_i: Uncertainty of task T_i near δ_{VOID}.

Step Size Constraint:

$$h_i = \max(\delta_{\text{VOID}}, h_{\text{optimal}}(T_i))$$

Ensures $h_i \geq \delta_{\text{VOID}}$.

Task Assignment Constraint:

$$\sum_{T_i \in \mathcal{T}_j} \lambda_i \leq C_j \quad \forall H_j \in \mathcal{H}$$

Where $\mathcal{T}_j \subseteq \mathcal{T}$ is the subset assigned to H_j.

8.1.2 Probabilistic Load Balancing

Objective: Distribute workloads based on computational uncertainties, allocating resources to tasks appropriately.

Probability Distribution for Assignment:

$$P(H_j|T_i) = \frac{C_j \cdot e^{-U_i}}{\sum_{k=1}^{n} C_k \cdot e^{-U_i}}$$

Ensures:

$$\sum_{j=1}^{n} P(H_j|T_i) = 1 \quad \forall T_i \in \mathcal{T}$$

Optimization Goal: Maximize overall computational efficiency η:

$$\eta = \sum_{i=1}^{m} \sum_{j=1}^{n} P(H_j|T_i) \cdot \text{Efficiency}(T_i, H_j)$$

Subject to step size and load constraints.

8.1.3 Assumptions and Limitations

Assumptions:

1. Finite Precision: Computations use finite precision with δ_{VOID} as the minimal step size.
2. Independent Tasks: Tasks are independent, requiring no synchronization beyond initial assignment.
3. Uniform Hardware Capabilities: Hardware units have similar performance characteristics for uniform task distribution.

Limitations:

1. Scalability Constraints: Increased hardware units may introduce communication overhead.
2. Dynamic Workloads: Interdependent or highly dynamic tasks may need advanced synchronization not covered here.
3. Probabilistic Accuracy: Variability in task assignments may affect execution consistency, necessitating robust error handling.

8.1.4 Convergence Properties of Probabilistic Task Assignment

Theorem: Under finite precision and independent task processing, the probabilistic task assignment algorithm converges to an optimal distribution where the expected computational load on each hardware unit H_j is proportional to its capacity C_j.

Proof Sketch:

1. Initialization: Assign $P(H_j|T_i)$ based on C_j and U_i.
2. Assignment: Tasks are probabilistically assigned according to $P(H_j|T_i)$.
3. Expectation Calculation:

$$\mathbb{E}\left[\sum_{T_i \in \mathcal{T}_j} \lambda_i\right] = \sum_{T_i \in \mathcal{T}} P(H_j|T_i) \cdot \lambda_i$$

4. Proportionality: Assuming uniform distribution of λ_i and U_i, the expected load scales with C_j.

5. Convergence: With large m and n, the law of large numbers ensures actual load approaches expected load, achieving proportional distribution.

Implications:

- Load Balance: Even distribution minimizes idle times and maximizes resource utilization.
- Precision Constraints: Maintains $h \geq \delta_{\text{VOID}}$, ensuring computational efficiency.
- Stability: Reduced variance in task assignments leads to predictable performance.

8.1.5 Sample Pseudo-Code Snippets

```
# Function to compute VOID-constrained derivative in parallel
function parallel_void_derivative(f, x_values, delta_void, num_cores):
    function worker(x):
        h_optimal = compute_optimal_h(f, x)
        h = max(delta_void, h_optimal)
        return (f(x + h) - f(x)) / h

    partitions = partition(x_values, num_cores)
    derivatives = parallel_map(worker, partitions, num_cores)
    return flatten(derivatives)

# Helper to compute optimal step size
function compute_optimal_h(f, x):
    # Error estimation logic
    return h_optimal
```

Explanation:

1. `compute_optimal_h`: Determines h_{optimal} based on local error estimates.
2. `worker`: Calculates the derivative ensuring $h \geq \delta_{\text{VOID}}$.
3. `parallel_map`: Distributes tasks across cores.
4. `flatten`: Aggregates results.

8.1.6 Probabilistic Load Balancing in Numerical Integration

```
# Function for VOID-constrained numerical integration with probabilistic load balancing
function parallel_void_integration(f, a, b, delta_void, num_cores):
    sub_intervals = divide_interval(a, b, delta_void)
```

```
    weights = [assess_complexity(f, interval) for interval in sub_intervals]
    probabilities = normalize(weights)
    assignments = probabilistic_assign(sub_intervals, probabilities, num_cores)

    function worker(interval):
        h = max(delta_void, compute_optimal_h(f, interval))
        return trapezoidal_rule(f, interval.a, interval.b, h)

    partial_integrals = parallel_map(worker, assignments, num_cores)
    return sum(partial_integrals)

# Helper to assess complexity
function assess_complexity(f, interval):
    # Assess based on derivative or function behavior
    return complexity_score

# Helper to assign intervals based on probabilities
function probabilistic_assign(sub_intervals, probabilities, num_cores):
    assignments = [[] for _ in range(num_cores)]
    for interval, prob in zip(sub_intervals, probabilities):
        core = select_core_based_on_probability(prob, num_cores)
        assignments[core].append(interval)
    return assignments

# Helper to select core
function select_core_based_on_probability(prob, num_cores):
    # Weighted random selection
    return selected_core
```

Explanation:

1. `divide_interval`: Splits $[a, b]$ ensuring $h \geq \delta_{\text{VOID}}$.
2. `assess_complexity`: Evaluates computational intensity of each sub-interval.
3. `probabilistic_assign`: Assigns sub-intervals to cores based on probabilities.
4. `worker`: Performs trapezoidal integration on assigned intervals.
5. `parallel_map`: Executes integrations in parallel.
6. `sum`: Aggregates results for total integral.

8.1.7 Practical Guidance for Implementation

- Step Size Management: Ensure $h \geq \delta_{\text{VOID}}$ to maintain efficiency and precision.

- Load Distribution: Use probabilistic methods to balance load based on task complexity and hardware capacity.
- Parallel Execution: Utilize parallel processing frameworks (e.g., OpenMP, MPI, CUDA) to implement `parallel_map`.
- Error Handling: Implement robust strategies to manage variability from probabilistic assignments.
- Performance Monitoring: Continuously monitor and adjust load balancing to optimize resource utilization.

9 Axiom 9.8: VOID-Constrained Error Handling in Hardware

VOID-constrained error handling ensures that floating-point precision errors near δ_{VOID} are managed effectively, acknowledging probabilistic uncertainties.

9.1 Error Handling Strategy

- Error Thresholding: Errors smaller than δ_{VOID} are considered probabilistically insignificant.
- Probabilistic Error Models: Errors are treated as random variables with distributions reflecting hardware limitations and VOID Granularity.

9.2 Practical Application

In sensor-based systems (e.g., telescopes, microscopes), where high precision is critical but hardware constraints and measurement uncertainties introduce noise, this method stabilizes computations by managing inevitable errors probabilistically.

10 Additional Concepts

10.1 Axiom 9.9: VOID-Constrained Iterative Methods

Iterative numerical methods (e.g., Newton-Raphson, gradient descent) are adjusted to stop refining solutions beyond δ_{VOID}, considering probabilistic indistinguishability.

10.2 Termination Criterion

An iterative method converges when:

$$|x_{n+1} - x_n| \leq \delta_{\text{VOID}}$$

At this point, further iterations are probabilistically indistinct and do not yield meaningful improvements.

10.3 Practical Application

In optimization problems and machine learning training, this prevents overfitting and unnecessary computations by acknowledging when further refinement falls below meaningful precision due to probabilistic uncertainties.

11 Conclusion

VOID-constrained numerical methods provide a structured framework for addressing the practical and theoretical constraints of modern computational problems, incorporating probabilistic uncertainties inherent at scales below δ_{VOID}. By integrating adaptive methods, hardware-aware algorithms, and probabilistic error management, these methods enhance both the efficiency and accuracy of simulations in fields as diverse as physics, engineering, and artificial intelligence.

Recognizing and respecting the limits of precision imposed by hardware, multi-scale behaviors, and inherent uncertainties, VOID-constrained methods offer robust solutions for real-world applications where infinite precision is neither feasible nor meaningful. This approach aligns computational practices with the principles of the VOID Granularity Framework, ensuring that numerical methods are both practically efficient and theoretically sound.

12 List of axioms

1. Axiom 9.1: VOID-Constrained Numerical Differentiation
2. Axiom 9.2: VOID-Constrained Numerical Integration
3. Axiom 9.3: Adaptive VOID Integration
4. Axiom 9.4: Hardware-Constrained VOID Threshold
5. Axiom 9.5: Adaptive Methods for Multi-Scale Problems
6. Axiom 9.6: Parallelization and Hardware-Aware VOID Algorithms
7. Axiom 9.7: Parallelization and Hardware-Aware VOID Algorithms (duplicate)
8. Axiom 9.8: VOID-Constrained Error Handling in Hardware
9. Axiom 9.9: VOID-Constrained Iterative Methods