

JavaScript Integration Library

User Manual

Version 3.01

A JavaScript Class for Numerical Integration

By Henrik Vestermark (hve@hvks.com)

January 2026

Contents

1. Introduction.....	4
1.1 Features.....	4
1.2 Supported Integration Methods.....	4
2. Quick Start.....	6
2.1 Basic Usage.....	6
2.2 Accessing Diagnostics.....	6
2.3 Verbose Mode.....	6
3. Class Reference.....	7
3.1 Constructor.....	7
3.2 Properties.....	7
3.3 Methods.....	7
3.4 Utility Methods.....	11
4. Usage Examples.....	12
4.1 Basic Integration.....	12
4.2 Comparing Methods.....	12
4.3 Fox-Romberg for Algebraic Singularities.....	12
4.4 Difficult Integrand with Gauss-Kronrod.....	12
4.5 Fixed-Step Integration.....	12
4.6 Using Function Objects.....	12
4.7 Analyzing Convergence.....	13
5. Advanced Topics.....	14
5.1 Understanding Convergence Power (q).....	14
5.2 Choosing the Right Method.....	14
5.3 Gauss-Kronrod Adaptive Strategy.....	14
5.4 Expression String Features.....	14
5.5 Performance Considerations.....	15
6. Error Handling.....	16
6.1 Invalid Expressions.....	16
6.2 Function Evaluation Errors.....	16
6.3 Convergence Warnings.....	16
7. API Reference Summary.....	17
7.1 Integration Methods.....	17
7.2 Properties.....	17

- 7.3 Utility Methods 17
- 8. Version History 18
- 9. Mathematical Background 19
 - 9.1 Fox-Romberg Integration..... 19
 - 9.2 Gauss-Kronrod Rules..... 19
 - 9.3 Gauss-Legendre Quadrature 19
 - 9.4 Romberg Integration 19
 - 9.5 Simpson's Rule..... 19
- 10. License and Contact..... 20
 - 10.1 Copyright 20
 - 10.2 Contact Information 20
 - 10.3 Acknowledgments..... 20

1. Introduction

The Integration library provides a comprehensive JavaScript class for numerical integration using multiple methods. It supports both fixed-step and adaptive integration techniques with detailed convergence diagnostics.

This library is designed for web applications, scientific computing, and educational purposes that require accurate numerical integration. All methods provide iteration history, error estimates, and counts of function evaluations for performance analysis.

1.1 Features

- Eight integration methods: Fox-Romberg, Gauss-Chebyshev, Gauss-Kronrod, Gauss-Legendre, Midpoint, Romberg, Simpson, and Trapezoid
- Both adaptive and fixed-step integration modes
- Automatic convergence detection with power estimation
- Detailed iteration history and diagnostics
- Function evaluation counter for performance analysis
- Support for both function objects and string expressions
- Optional verbose mode for detailed integration progress

1.2 Supported Integration Methods

Fox-Romberg Integration

Adaptive Romberg variant that detects fractional convergence powers ($q=0.5$ or $q=1.5$) and adjusts the extrapolation accordingly. Ideal for integrands with algebraic singularities.

Gauss-Chebyshev Quadrature

Specialized for integrands with weight function $1/\sqrt{1-x^2}$. Excellent for oscillatory functions over $[-1,1]$.

Gauss-Kronrod Quadrature (Adaptive)

Adaptive GK(7,15) quadrature with automatic subdivision. Most sophisticated method, ideal for difficult integrands with discontinuities or rapid variations.

Gauss-Legendre Quadrature

High-precision integration using Gauss-Legendre nodes and weights. Optimal for polynomials and smooth functions.

Midpoint Rule

Composite midpoint rule. Useful for integrands with endpoint singularities.

Romberg Integration

Richardson extrapolation on the trapezoid rule. Excellent for smooth functions, achieving high accuracy with fewer evaluations.

Simpson's Rule

Composite Simpson's rule with adaptive refinement. Provides 4th-order accuracy for smooth functions.

Trapezoid Rule

Composite trapezoidal rule with iterative refinement. Simple and reliable for general use.

2. Quick Start

2.1 Basic Usage

Create an Integration object with a function or expression string:

```
// Using a function const integ = new Integration(x => Math.sin(x));  
// Using an expression string const integ2 = new Integration("sin(x)");
```

Compute the integral using your preferred method:

```
const result = integ.simpson(0, Math.PI); console.log(result); // 2.0  
(approximately)
```

2.2 Accessing Diagnostics

After integration, access convergence information:

```
console.log("Method:", integ.method()); console.log("Function  
evaluations:", integ.cnt); console.log("Iterations:", integ.i.length);  
console.log("Convergence powers:", integ.q); console.log("Accuracy:",  
integ.accuracy);
```

2.3 Verbose Mode

Enable detailed diagnostics during integration:

```
const integ = new Integration("exp(-x*x)", { verbose: true }); const  
result = integ.gaussKronrod(0, 1); const diagnostics =  
integ.getVerbose(); console.log(diagnostics.log); // Detailed  
iteration log
```

3. Class Reference

3.1 Constructor

```
new Integration(eqOrFunc, options = {})
```

Parameters:

- **eqOrFunc**: Function ($x \Rightarrow f(x)$) or expression string (e.g., `"sin(x) * exp(-x)"`)
- **options**: Optional configuration object. Currently supports `{ verbose: true/false }`

Expression String Support:

Expression strings are automatically compiled into functions. Supported operations include all standard Math functions (sin, cos, exp, log, sqrt, etc.) and constants (PI, E, etc.). Use `^` for exponentiation, which is converted to the JavaScript `**` operator.

3.2 Properties

eq : *string* — String representation of the function (original expression or "(function)")

cnt : *number* — Total number of function evaluations

i : *number[]* — Array of integral estimates from each iteration

q : *number[]* — Array of convergence power estimates (base-2 logarithm of error reduction ratios)

intgmethod : *string* — Name of the last integration method used

accuracy : *string* — String representation of estimated accuracy (e.g., "1e-10")

verbose : *object* — Verbose diagnostics configuration and log storage

3.3 Methods

foxromberg(a, b, maxiter)

Fox-Romberg integration with adaptive convergence power estimation.

Parameters:

- **a**: Lower integration bound
- **b**: Upper integration bound
- **maxiter**: (Optional) Maximum Romberg iterations (default: 15)

Fox-Romberg performs a trial Romberg run to detect non-integer convergence powers (typically $q=0.5$ or $q=1.5$), then reruns with a modified first extrapolation step that uses 2^q instead of the standard factor of 4. This is particularly effective for integrands with algebraic singularities or other non-smooth behavior where the standard Romberg method shows fractional convergence rates.

```
// Automatic q detection and adaptation const result =
integ.foxromberg(0, 1, 15);
```

gaussChebyshev(a, b, n)

Gauss-Chebyshev quadrature for integrals with weight function $1/\sqrt{1-x^2}$.

Parameters:

- **a**: Lower integration bound (typically -1)
- **b**: Upper integration bound (typically 1)
- **n**: Number of Gauss-Chebyshev points (default: 64)

Note: Specialized for integrals of the form:

```
∫[a,b] f(x)/√(1-x2) dx  const result = integ.gaussChebyshev(-1, 1, 64);
```

gaussKronrod(a, b, options)

Adaptive Gauss-Kronrod GK(7,15) quadrature with automatic interval subdivision.

Parameters:

- **a**: Lower integration bound
- **b**: Upper integration bound
- **options**: (Optional) Configuration object

Options object:

- **rtol**: Relative tolerance (default: 1e-10)
- **atol**: Absolute tolerance (default: 1e-12)
- **maxEvals**: Maximum function evaluations (default: 100000)
- **maxIntervals**: Maximum subdivisions (default: 5000)
- **minInterval**: Minimum interval width (default: 0, disabled)
- **errSafety**: Error estimate safety factor (default: 1.1)

Gauss-Kronrod is the most sophisticated method, using adaptive subdivision to handle difficult integrands. It automatically focuses computational effort on regions where the integrand varies most rapidly. The error estimate is $|K_{15} - G_7|$, where K_{15} is the 15-point Kronrod rule, and G_7 is the embedded 7-point Gauss rule.

```
// Default settings const result = integ.gaussKronrod(0, 1); // Custom
tolerance const result2 = integ.gaussKronrod(0, 1, {  rtol: 1e-12,
atol: 1e-14,   maxEvals: 50000 });
```

gaussLegendre(a, b, n)

Gauss-Legendre quadrature integration.

Parameters:

- **a**: Lower integration bound
- **b**: Upper integration bound
- **n**: Number of Gauss points (2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 64, or 100). Default: 64

Gauss-Legendre quadrature is optimal for integrating polynomials. An n-point rule exactly integrates polynomials of degree $2n-1$ or less.

```
const result = integ.gaussLegendre(0, 1, 64);
```

midpoint(a, b, n)

Composite midpoint rule integration.

Parameters and Returns: Same as `simpson()`

The midpoint rule is particularly useful for integrands with endpoint singularities since it never evaluates the function at the endpoints.

```
const result = integ.midpoint(0, 1);
```

romberg(a, b, maxiter)

Romberg integration using Richardson extrapolation on the trapezoid rule.

Parameters:

- **a**: Lower integration bound
- **b**: Upper integration bound
- **maxiter**: (Optional) Maximum Romberg iterations (default: 20)

Romberg integration builds a triangular tableau of increasingly accurate estimates using polynomial extrapolation. Particularly effective for smooth integrands.

```
const result = integ.romberg(0, 1, 15);
```

simpson(a, b, n)

Composite Simpson's rule integration with adaptive or fixed-step mode.

Parameters:

- **a**: Lower integration bound
- **b**: Upper integration bound
- **n**: (Optional) Number of Simpson panels. If provided, performs single fixed-step calculation without iteration.

Returns: Integral estimate

Behavior:

- Without n: Iterates with $n = 1, 2, 4, 8, \dots$ until convergence (max 32768 panels)
- With n: Performs single calculation, no convergence checking

```
const result = integ.simpson(0, 1); // Adaptive const result2 =  
integ.simpson(0, 1, 100); // Fixed 100 panels
```

trapezoid(a, b, n)

Composite trapezoid rule integration with adaptive or fixed-step mode.

Parameters and Returns: Same as `simpson()`

```
const result = integ.trapezoid(0, 1);
```

3.4 Utility Methods

reset()

Clears iteration history, convergence estimates, and resets the function evaluation counter. Called automatically at the start of each integration.

method(m)

Get or set the integration method name. With no arguments, returns the current method. With argument, sets method name (string).

value(x)

Evaluates the function at x and increments the evaluation counter. Handles errors gracefully.

getVerbose()

Returns a detailed diagnostics object containing:

- **method**: Integration method name
- **eq**: Function expression
- **accuracy**: Accuracy estimate
- **evals**: Total function evaluations
- **iterations**: Number of iterations
- **i**: Array of iteration estimates
- **q**: Array of convergence powers
- **log**: Method-specific detailed log
- **adp_log**: (Gauss-Kronrod only) Adaptive subdivision log

toString()

Returns string representation of the function (expression or "(function)").

valueOf()

Returns the compiled function object itself.

4. Usage Examples

4.1 Basic Integration

```
// Integrate sin(x) from 0 to π const integ = new
Integration("sin(x)"); const result = integ.simpson(0, Math.PI);
console.log(result); // ≈ 2.0 console.log("Evaluations:", integ.cnt);
console.log("Iterations:", integ.i.length);
```

4.2 Comparing Methods

```
const f = "exp(-x*x)"; const [a, b] = [0, 1]; const methods = [
  { name: "Simpson", fn: (i) => i.simpson(a, b) }, { name: "Romberg", fn:
  (i) => i.romberg(a, b) }, { name: "Fox-Romberg", fn: (i) =>
  i.foxromberg(a, b) }, { name: "Gauss-Legendre", fn: (i) =>
  i.gaussLegendre(a, b, 64) }, { name: "Gauss-Kronrod", fn: (i) =>
  i.gaussKronrod(a, b) } ]; methods.forEach(m => { const integ = new
  Integration(f); const result = m.fn(integ); console.log(`${m.name}:
  ${result.toFixed(15)}`); console.log(` Evals: ${integ.cnt}, Iters:
  ${integ.i.length}`); });
```

4.3 Fox-Romberg for Algebraic Singularities

```
// Integrate sqrt(x) from 0 to 1 // Has fractional convergence due to
singularity at x=0 const integ = new Integration("sqrt(x)", { verbose:
true }); const result = integ.foxromberg(0, 1); console.log("Result:",
result); // Exact: 2/3 console.log("Accuracy:", integ.accuracy);
const diag = integ.getVerbose(); // Check if Fox-Romberg detected q
const foxLog = diag.log.find(l => l.qFox); if (foxLog) {
console.log("Detected convergence power q =", foxLog.qFox); }
```

4.4 Difficult Integrand with Gauss-Kronrod

```
// Function with sharp peak const integ = new Integration( x =>
Math.exp(-100 * (x - 0.5) ** 2), { verbose: true } ); const result =
integ.gaussKronrod(0, 1, { rtol: 1e-10, atol: 1e-12 });
console.log("Result:", result); console.log("Accuracy:",
integ.accuracy); console.log("Total evaluations:", integ.cnt); const
diag = integ.getVerbose(); console.log("Intervals subdivided:",
diag.adp_log.length);
```

4.5 Fixed-Step Integration

```
// Use exactly 1000 Simpson panels const integ = new Integration("1/(1
+ x^2)"); const result = integ.simpson(0, 1, 1000);
console.log(result); // ≈ 0.785398 (π/4)
```

4.6 Using Function Objects

```
// Define complex function function myFunc(x) { if (x === 0) return
1; return Math.sin(x) / x; // sinc function } const integ = new
Integration(myFunc); const result = integ.romberg(0, 10);
console.log("Integral of sinc(x):", result);
```

4.7 Analyzing Convergence

```
const integ = new Integration("sqrt(x)"); integ.simpson(0, 1);
console.log("Iteration history:"); integ.i.forEach((val, idx) => {
const q = integ.q[idx - 1]; console.log(` Iter ${idx}:
${val.toExponential(15)} ` + ` (q=${q !== undefined ?
q.toFixed(2) : "N/A"})`); }); // q ≈ 4.0 indicates 4th-order
convergence (Simpson's rule)
```

5. Advanced Topics

5.1 Understanding Convergence Power (q)

The convergence power q measures how rapidly the method converges. It is computed from successive differences in the iteration history:

$$q = \log_2(|\Delta_{i-1}| / |\Delta_i|) \quad \text{where } \Delta_i = I_i - I_{i-1}$$

Interpretation:

- **$q \approx 2.0$:** Second-order convergence (trapezoid, midpoint)
- **$q \approx 4.0$:** Fourth-order convergence (Simpson)
- **$q > 4.0$:** Higher-order convergence (Romberg, Gauss methods)
- **$q < 1.0$:** Slow or non-convergence (integration may be difficult)

5.2 Choosing the Right Method

Method	Best For	Typical q
Fox-Romberg	Algebraic singularities, fractional convergence	~0.5 or ~1.5
Gauss-Chebyshev	Oscillatory functions on [-1,1]	High
Gauss-Kronrod	Difficult integrands, adaptive needs	Adaptive
Gauss-Legendre	Polynomials, smooth functions	High
Midpoint	Endpoint singularities	~2.0
Romberg	Smooth functions, high accuracy needs	~8.0+
Simpson	General-purpose, smooth functions	~4.0
Trapezoid	Simple problems, quick estimates	~2.0

5.3 Gauss-Kronrod Adaptive Strategy

The Gauss-Kronrod method uses a sophisticated adaptive algorithm:

- Maintains a priority queue of intervals ordered by error estimate
- Always subdivides the interval with the largest estimated error
- Uses embedded rules GK(7,15) for error estimation without extra evaluations
- Terminates when the global error estimate satisfies the tolerance
- Provides detailed subdivision log in verbose mode

5.4 Expression String Features

Expression strings support:

- **Math functions:** abs, sin, cos, tan, exp, log, sqrt, etc.
- **Constants:** PI, E, LN2, LN10, SQRT2, etc.
- **Operators:** +, -, *, /, ^ (exponentiation)
- **Security:** Expressions are validated to prevent code injection

```
const valid = [ "sin(PI*x)", "exp(-x^2)", "1/(1 + x*x)",
"sqrt(abs(x)) * log(x + 1)" ]; // These will throw errors:
invalid = [ "alert('hack')", // No function calls except Math "x;
```

```
y = 5", // No semicolons "while(true){}", // No control  
structures ];
```

5.5 Performance Considerations

- **Function compilation:** Expression strings are compiled once in the constructor
- **Reusability:** Create one Integration object and call multiple methods
- **Evaluation counting:** Use the cnt property to compare method efficiency
- **Fixed-step mode:** Faster for known panel counts, no convergence overhead
- **Verbose mode:** Adds minimal overhead, safe to use in production

6. Error Handling

6.1 Invalid Expressions

The constructor throws an error if the expression contains illegal tokens:

```
try { const integ = new Integration("x; alert(1)"); } catch (e) {
  console.error(e.message); // "Illegal tokens in expression." }
```

6.2 Function Evaluation Errors

If the function throws during evaluation, an alert is shown and NaN is returned:

```
const integ = new Integration(x => { if (x < 0) throw new
  Error("Negative x"); return Math.sqrt(x); }); // This will alert the
error and return NaN const result = integ.simpson(-1, 1);
```

6.3 Convergence Warnings

Methods may reach maximum iterations without full convergence. Check the convergence power q and iteration history to diagnose:

```
const integ = new Integration("1/x"); // Singularity at 0 const result
= integ.simpson(0, 1); // Difficult integral if
(integ.q[integ.q.length - 1] < 1.0) { console.warn("Slow convergence
detected!"); console.log("Last q value:", integ.q[integ.q.length -
1]); }
```

7. API Reference Summary

7.1 Integration Methods

Method	Parameters	Mode
foxromberg(a, b, maxiter)	a, b: bounds maxiter: max iterations	Adaptive with q detection
gaussChebyshev(a, b, n)	a, b: bounds n: Gauss points	Fixed
gaussKronrod(a, b, opts)	a, b: bounds, opts: tolerance, config	Adaptive
gaussLegendre(a, b, n)	a, b: bounds n: Gauss points	Fixed
midpoint(a, b, n)	a, b: bounds n: optional panels	Adaptive or fixed
romberg(a, b, maxiter)	a, b: bounds maxiter: max iterations	Adaptive
simpson(a, b, n)	a, b: bounds n: optional panels	Adaptive or fixed
trapezoid(a, b, n)	a, b: bounds n: optional panels	Adaptive or fixed

7.2 Properties

Property	Type	Description
eq	string	Function expression or "(function)"
cnt	number	Function evaluation count
i	number[]	Iteration history
q	number[]	Convergence power estimates
intgmethod	string	Current method name
accuracy	string	Estimated accuracy (e.g., "1e-10")
verbose	object	Verbose diagnostics configuration

7.3 Utility Methods

Method	Returns	Description
reset()	void	Clear iteration history
method(m)	string	Get/set method name
value(x)	number	Evaluate the function at x
getVerbose()	object	Get detailed diagnostics
toString()	string	Get expression string
valueOf()	function	Get function object

8. Version History

Version	Date	Changes
1.01	Apr 2007	Initial release
2.03	May 2007	Added Gauss-Legendre and Gauss-Chebyshev
2.04	May 2007	Added Gauss-Hermite
2.05	Jun 2007	Corrected bug in q calculation
2.06	Nov 2009	Added the Double Exponential method
2.07	May 2011	Fixed premature iteration stops
2.08	Jan 2026	Fixed Gauss-Chebyshev bug, added Gauss-Kronrod, and removed Gauss-Hermite
3.01	Jan 2026	Complete rewrite using JavaScript class structure, added verbose mode

9. Mathematical Background

9.1 Fox-Romberg Integration

Fox-Romberg adapts Romberg integration for integrands with fractional convergence powers:

Trial phase: Detect q from convergence pattern (typically 0.5 or 1.5)
 Main phase: Modified first extrapolation $R(n,1) = R(n,0) + (R(n,0) - R(n-1,0)) / (2^q - 1)$

This modification allows proper extrapolation for integrands like \sqrt{x} or functions with algebraic endpoint singularities, which show fractional convergence in standard Romberg. The method automatically falls back to standard Romberg if no fractional power is detected.

9.2 Gauss-Kronrod Rules

GK(7,15) uses a 15-point Kronrod rule with an embedded 7-point Gauss rule:

Error estimate $\approx |K_{15} - G_7|$ K_{15} : 15-point Kronrod estimate G_7 : 7-point Gauss estimate (subset of Kronrod points)

This provides an error estimate without additional function evaluations. The adaptive algorithm subdivides intervals with large estimated errors until the global error tolerance is satisfied.

9.3 Gauss-Legendre Quadrature

Uses optimally chosen points x_i (roots of Legendre polynomials) and weights w_i :

$$\int_{[-1,1]} f(x) dx \approx \sum w_i f(x_i)$$

An n -point Gauss-Legendre rule exactly integrates polynomials of degree $2n-1$ or less. For arbitrary intervals $[a,b]$, a linear transformation is applied.

9.4 Romberg Integration

Romberg integration applies Richardson extrapolation to trapezoid estimates:

$$R(n,m) = (4^m R(n,m-1) - R(n-1,m-1)) / (4^m - 1)$$

Each extrapolation level increases the order of accuracy by 2. The diagonal elements $R(n,n)$ converge very rapidly for smooth integrands.

9.5 Simpson's Rule

Simpson's rule approximates the integrand using parabolic segments:

$$\int [a,b] f(x) dx \approx (h/3) [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + f(x_n)]$$

Error term: $O(h^4)$, where $h = (b-a)/n$. This fourth-order accuracy makes Simpson's rule very efficient for smooth functions.

10. License and Contact

10.1 Copyright

Copyright © 2007-2026 Henrik Vestermark. All Rights Reserved.

This software is subject to the terms and conditions of the Henrik Vestermark Software License Agreement.

10.2 Contact Information

Henrik Vestermark

Email: hve@hvks.com

Website: <https://hvks.com>

10.3 Acknowledgments

This library builds upon decades of research in numerical quadrature. The Gauss-Kronrod implementation follows the design principles established by QUADPACK and similar scientific libraries. Thanks to the numerical analysis community for their continued contributions to computational mathematics.