User Manual
Numerical Analysis Calculator

November 2025

Contents

1 Introduction

1.1 What is the Numerical Analysis Calculator?

The Numerical Analysis Calculator is a professional web application designed to solve complex mathematical problems using numerical methods. It implements **26 different methods** organized into three main categories:

- **Nonlinear Equations** (7 methods)
- **Linear Systems** (13 methods)
- **Interpolation** (6 methods)

1.2 Main Features

Intuitive Interface: Modern design inspired by professional calculators

Interactive Graphics: Function visualization with Desmos API

Detailed Results: Step-by-step iteration tables

Responsive: Works on PC, tablets and smartphones

No Installation: 100% online web application

Intelligent Validation: Educational error messages

1.3 Who is This Application For?

- Engineering and exact sciences students
- Mathematics and numerical analysis professors
- Engineers who need to solve numerical problems
- Researchers in scientific areas
- Anyone interested in numerical methods

2 Application Access

2.1 System Requirements

Supported Browsers:

- Google Chrome 90+ (Recommended)
- Mozilla Firefox 88+
- Safari 14+
- Microsoft Edge 90+

Internet Connection:

- Required to load the application and Desmos graphics

Devices:

- Desktop computers (Windows, Mac, Linux)
- Laptops
- Tablets (iPad, Android)
- Smartphones (iOS, Android)

2.2 How to Access

1. Open your web browser
2. Enter the application URL
3. The application will load automatically
4. No registration or login required

3 User Interface

3.1 Main Components

3.1.1 Header

- **Logo**: Σ (Sigma) identifying symbol
- **Title**: "Numerical Calculator"
- **Description**: Information about the application
- **Available Functions**: List of supported mathematical functions

3.1.2 Methods Panel (Left)

Contains four main tabs:

- **Equation Solving**: Equation methods
- **Linear Systems**: Linear equation systems
- **Interpolation**: Interpolation methods
- **Additional Methods**: Additional methods

3.1.3 Results Panel (Right)

- **Graph**: Function visualization (when applicable)
- **Results**: Detailed output with iteration tables
- **Warnings**: Suggestions and informative messages

3.2 Tab Navigation

To switch between categories:

1. Click on the desired tab
2. Available methods will be displayed automatically
3. Select a method by clicking on its card

   Visual indicators:

- Active tab: Blue background and bottom line
- Selected method: Blue border and highlighted background

3.3 Parameter Form

Once a method is selected, a dynamic form will appear with the necessary parameters:

- **Text fields**: For mathematical functions
- **Numeric fields**: For initial values, tolerances, etc.
- **Text areas**: For matrices and vectors
- **Calculate button**: Executes the selected method

4 Nonlinear Equation Methods

Nonlinear equation methods are used to find roots (values of x where $f(x) = 0$) of functions.

4.1 Incremental Search

Description:
Searches for intervals where the function changes sign, indicating the presence of a root.

When to Use It:

- To explore a function and find all its roots
- When you don't know where the roots are
- As a preliminary step to other methods

Parameters:

- **Function** $f(x)$: Function to analyze
- x_0: Starting search point
- Δ **(Delta)**: Increment (can be positive or negative)
- **Nmax**: Maximum number of iterations

Usage Example:

```
Function: x**3 - x - 2
x0: 0
Delta: 0.5
Nmax: 100
```

Results:

- List of intervals $[a, b]$ where there is a sign change
- Table with values of x and $f(x)$ at each iteration

Advantages:

- Simple to understand
- Finds multiple roots
- Does not require derivatives

Disadvantages:

- Does not find multiple roots (where $f(x)$ touches the axis but does not cross)
- Can be slow
- Depends on the value of Δ

4.2 Bisection

Description:
Repeatedly divides an interval in half until finding the root with desired precision.

When to Use It:

- When you have an interval $[a, b]$ where $f(a)$ and $f(b)$ have opposite signs
- You need guaranteed convergence
- The function is continuous

Parameters:

- **Function** $f(x)$: Function to solve
- a: Left endpoint of interval
- b: Right endpoint of interval
- **Tolerance**: Desired precision (e.g., 10^{-6})
- **Nmax**: Maximum iterations

Usage Example:

```
Function: x**3 - x - 2
a: 1
b: 2
Tolerance: 0.000001
Nmax: 50
```

Results:

- Iteration table with:
  - iter: Iteration number
  - a: Current left endpoint
  - x_m: Midpoint
  - b: Current right endpoint
  - $f(x_m)$: Function value at x_m
  - E: Absolute error
- Final approximate root

Advantages:

- Always converges (if there is a root in the interval)
- Very robust
- Easy to implement

Disadvantages:

- Slow convergence
- Requires initial interval with sign change
- Does not work for multiple roots

Interpretation:

- If $E <$ Tolerance: Root found with desired precision
- If reached Nmax: Increase Nmax or reduce tolerance

4.3 False Position

Description:
Similar to bisection but uses linear interpolation instead of midpoint.

When to Use It:

- When you have an interval with sign change
- You want faster convergence than bisection
- The function is approximately linear in the interval

Parameters:

- **Function** $f(x)$: Target function
- a: Left endpoint
- b: Right endpoint
- **Tolerance**: Precision
- **Nmax**: Maximum iterations

Usage Example:

```
Function: cos(x) - x
a: 0
b: 1
Tolerance: 1e-7
Nmax: 100
```

Results:

- Table similar to bisection
- The point x_m is calculated with the formula:

$$x_m = \frac{f(b) \cdot a - f(a) \cdot b}{f(b) - f(a)}$$

Advantages:

- Faster than bisection in many cases
- Guarantees convergence

Disadvantages:

- Can converge very slowly if $f(a)$ and $f(b)$ are very different
- Can "stagnate" at one endpoint

4.4 Fixed Point

Description:
Finds a value x such that $x = g(x)$, transforming $f(x) = 0$ into $x = g(x)$.

When to Use It:

- When you can rewrite $f(x) = 0$ as $x = g(x)$
- For equations that naturally have fixed-point form

Parameters:

- **Function** $f(x)$: Original function (for graphing)
- **Function** $g(x)$: Fixed-point function ($x = g(x)$)
- x_0: Initial value
- **Tolerance**: Precision
- **Nmax**: Maximum iterations

Usage Example:

```
f(x): x**2 - 2
g(x): sqrt(2)  (or also: 2/x)
x0: 1.5
Tolerance: 1e-6
Nmax: 100
```

How to Create $g(x)$**:**
From $f(x) = x^2 - 2 = 0$:

- Option 1: $x^2 = 2 \rightarrow x = \sqrt{2} \rightarrow g(x) = \sqrt{2}$
- Option 2: $x = \frac{2}{x} \rightarrow g(x) = \frac{2}{x}$

Convergence Criterion:
For convergence, it must satisfy: $|g'(x)| < 1$ near the root

Results:

- Table with x_i, $g(x_i)$, $f(x_i)$ and error
- Approximate value of the root

Advantages:

- Simple to implement
- Does not require derivatives

Disadvantages:

- Does not always converge
- Critically depends on the choice of $g(x)$
- Can diverge with bad x_0

4.5 Newton-Raphson (Newton)

Description:
Uses the tangent to the function to approximate the root. Quadratic convergence.

When to Use It:

- When you have or can calculate $f'(x)$
- You need fast convergence
- You have a good initial estimate

Parameters:

- **Function** $f(x)$: Target function
- **Derivative** $f'(x)$: Derivative of f (optional, calculated numerically if not provided)
- x_0: Initial value
- **Tolerance**: Precision
- **Nmax**: Maximum iterations

Formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Usage Example:

```
f(x): x**3 - 2*x - 5
f'(x): 3*x**2 - 2
x0: 2
Tolerance: 1e-10
Nmax: 50
```

Results:

- Table with iter, x_i, $f(x_i)$, and error
- Typical convergence in 3-5 iterations

Advantages:

- Quadratic convergence (very fast)
- Requires few iterations

Disadvantages:

- Requires calculating $f'(x)$
- Can diverge if x_0 is poorly chosen
- Fails if $f'(x) = 0$

Tips:

- Choose x_0 close to the root
- If you know $f'(x)$ analytically, enter it for better precision
- If it diverges, try another x_0 or use bisection first

4.6 Secant

Description:
Approximation of Newton's method that does not require calculating derivatives.
When to Use It:

- When calculating $f'(x)$ is difficult or expensive
- You want fast convergence without derivatives
- You have two initial values close to the root

Parameters:

- **Function** $f(x)$: Target function
- x_0: First initial value
- x_1: Second initial value
- **Tolerance**: Precision
- **Nmax**: Maximum iterations

Formula:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Usage Example:

```
f(x): exp(x) - 3*x
x0: 0
x1: 1
Tolerance: 1e-8
Nmax: 50
```

Results:

- Iteration table
- Super-linear convergence (between linear and quadratic)

Advantages:

- Does not require derivatives
- Faster than fixed point
- Super-linear convergence

Disadvantages:

- Requires two initial values
- Can diverge if x_0 and x_1 are poorly chosen
- Slightly slower than Newton

4.7 Newton Multiple Roots

Description:
Variant of Newton's method designed for multiple roots (where $f(x) = f'(x) = 0$).

When to Use It:

- When you suspect the root has multiplicity > 1
- Standard Newton converges very slowly
- You have $f(x)$, $f'(x)$ and $f''(x)$

Parameters:

- **Function** $f(x)$: Target function
- **Derivative** $f'(x)$: First derivative
- **Second Derivative** $f''(x)$: Second derivative
- x_0: Initial value
- **Tolerance**: Precision
- **Nmax**: Maximum iterations

Formula:

$$x_{n+1} = x_n - \frac{f(x) \cdot f'(x)}{[f'(x)]^2 - f(x) \cdot f''(x)}$$

Usage Example:

```
f(x): (x - 2)**3
f'(x): 3*(x - 2)**2
f''(x): 6*(x - 2)
x0: 1.5
Tolerance: 1e-10
Nmax: 50
```

Multiple Roots:

- Multiplicity 2: $f(x) = (x - a)^2$
- Multiplicity 3: $f(x) = (x - a)^3$

Advantages:

- Quadratic convergence even for multiple roots
- Standard Newton only converges linearly for these roots

Disadvantages:

- Requires calculating $f''(x)$
- More complex to implement

5 Linear System Methods

Linear system methods solve systems of equations of the form $Ax = b$.

5.1 Input Format

Matrix A (coefficients):

```
4 -1 0
-1 4 -1
0 -1 4
```

Vector b (independent terms):

```
15 10 10
```

Format:

- Each row on a line
- Numbers separated by spaces
- No commas or brackets

5.2 Gaussian Elimination

Description:
Converts the matrix to upper triangular form through elementary operations.
When to Use It:

- For small to medium systems (up to ∼100 equations)
- When the matrix is not singular
- You need exact solution (without iterations)

Parameters:

- **Matrix A**: $n \times n$ coefficient matrix
- **Vector b**: Independent terms vector

Usage Example:

```
Matrix A:
2 1 -1
-3 -1 2
-2 1 2

Vector b:
8 -11 -3
```

Process:

1. **Forward elimination**: Creates zeros below the diagonal
2. **Back substitution**: Solves from the last equation

Results:

- Elimination stages (intermediate matrices)

- Solution vector x

Advantages:

- Direct method (non-iterative)
- Exact solution (within numerical precision)

Disadvantages:

- Can fail if there is zero pivot
- Numerically unstable without pivoting

5.3 Partial Pivoting

Description:
Gaussian Elimination with row interchange to improve stability.

When to Use It:

- When simple Gaussian fails
- To improve numerical stability
- As standard method for general systems

Difference with Simple Gaussian:

- At each step, searches for the element of largest absolute value in the column
- Exchanges rows to put that element on the diagonal

Example:

```
Matrix A:
0.0001 1
1 1

Vector b:
1 2
```

Without pivoting: Incorrect result
With pivoting: Correct result

Advantages:

- More numerically stable
- Prevents division by zero

Disadvantages:

- Slightly slower than simple Gaussian

5.4 Total Pivoting

Description:
Searches for the pivot in the entire submatrix (rows and columns).

When to Use It:

- Maximum numerical stability required
- Very ill-conditioned matrices
- When partial pivoting is not enough

Process:

- Searches for maximum element in the submatrix
- Exchanges rows AND columns
- More complex but more stable

Advantages:

- Maximum numerical stability

Disadvantages:

- Slower
- More complex (requires marking column permutations)

5.5 LU Decomposition

Description:
Decomposes A into product of two matrices: $A = LU$

- L: Lower triangular
- U: Upper triangular

When to Use It:

- You need to solve multiple systems with the same A
- Calculate determinants
- Matrix inversion

Advantages:

- Efficient for multiple b vectors
- Useful for additional calculations (determinant, inverse)

Process:

1. Decompose $A = LU$
2. Solve $Ly = b$ (forward substitution)
3. Solve $Ux = y$ (back substitution)

Variants:

5.5.1 LU Simple

- Without pivoting
- Can fail if there is zero pivot

5.5.2 LU Partial Pivot

- With partial pivoting: $PA = LU$
- More stable

5.5.3 Crout

- L has diagonal, U has 1s on diagonal
- Useful for certain algorithms

5.5.4 Doolittle

- L has 1s on diagonal, U has diagonal
- More common

5.6 Cholesky Decomposition

Description:
Special factorization for symmetric and positive definite matrices: $A = LL^T$

When to Use It:

- A is symmetric ($A_{ij} = A_{ji}$)
- A is positive definite (all eigenvalues > 0)
- Common in: least squares, finite elements

Advantages:

- Faster than LU (half the operations)
- More numerically stable
- Requires less memory

Disadvantages:

- Only for symmetric and positive definite matrices

Valid Matrix Example:

```
4 2 1
2 5 3
1 3 6
```

How to Verify if Positive Definite:

- All principal minors are positive
- Or: All eigenvalues are positive

5.7 Iterative Methods

Iterative methods start from an initial solution x_0 and refine it.

5.7.1 Jacobi

Description:
Solves each variable in terms of the others and updates simultaneously.

When to Use It:

- Large and sparse matrices
- Diagonally dominant
- Parallelization

Additional Parameters:

- x_0 **vector**: Initial solution (optional, default zeros)
- **Tolerance**: Precision
- **Nmax**: Maximum iterations

Convergence Criterion:

- Spectral radius of $T < 1$
- Diagonally dominant (sufficient but not necessary)

Diagonally Dominant:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

Example:

```
Matrix A:
4 -1 0
-1 4 -1
0 -1 4

Vector b:
15 10 10

x0: (optional)
Tolerance: 1e-6
Nmax: 100
```

Results:

- Iteration matrix T
- Vector C
- Spectral radius
- Iteration table
- Warning if may diverge

5.7.2 Gauss-Seidel

Description:
Similar to Jacobi but uses updated values immediately.

When to Use It:

- Same type of matrices as Jacobi
- Typically converges faster than Jacobi

Difference with Jacobi:

- Jacobi: Updates all variables simultaneously
- Gauss-Seidel: Updates variables sequentially using new values

Advantages over Jacobi:

- Faster convergence (typically)
- Requires less memory

Disadvantages:

- Not parallelizable

5.7.3 SOR (Successive Over-Relaxation)

Description:
Gauss-Seidel with relaxation factor ω to accelerate convergence.

Additional Parameters:

- **omega (ω)**: Relaxation factor ($0 < \omega < 2$)
  - $\omega < 1$: Under-relaxation (more stable)
  - $\omega = 1$: Standard Gauss-Seidel
  - $\omega > 1$: Over-relaxation (faster)

How to Choose ω:

- Theoretical: Optimal ω depends on spectral radius of Gauss-Seidel
- Practical: Try values between 1.0 and 1.9
- Typical: 1.5 is good starting point

Example:

```
omega: 1.5
```

Advantages:

- Can be much faster than Gauss-Seidel
- Useful for large matrices

Disadvantages:

- Choosing optimal ω can be difficult

5.8 Forward/Backward Substitution

Description:
Direct methods for triangular systems.

5.8.1 Forward Substitution

When to Use It:

- Lower triangular system $Lx = b$
- As step in LU factorization

Input format:
Augmented matrix $[L|b]$:

```
2 0 0 4
1 3 0 5
2 1 4 6
```

5.8.2 Backward Substitution

When to Use It:

- Upper triangular system $Ux = b$
- As step in LU factorization or Gaussian

Input format:

Augmented matrix $[U|b]$:

```
2 1 -1 8
0 3 2 -11
0 0 4 -3
```

6 Interpolation Methods

Interpolation constructs a function that passes through a given set of points.

6.1 Input Format

X Points (x coordinates):

```
0 1 2 3 4
```

Y Points (y coordinates):

```
1 2.5 3.2 4.1 5.5
```

Requirements:

- Same number of values in X and Y
- Minimum 2 points
- Points will be automatically sorted by X

6.2 Vandermonde

Description:
Constructs a polynomial $P(x)$ of degree $n-1$ that passes through n points.

When to Use It:

- Few points (< 10)
- You need the explicit polynomial
- Well-spaced points

Method:
Solves system $Vc = y$ where V is the Vandermonde matrix

Example:

```
X: 0 1 2
Y: 1 3 2
```

Results:

- Vandermonde matrix
- Polynomial coefficients $[a_0, a_1, a_2, \ldots]$
- Polynomial: $P(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$

Advantages:

- Obtains explicit polynomial
- Simple to understand

Disadvantages:

- Numerically unstable for many points
- Runge phenomenon (oscillations) with many points

Warning:
For more than 10 points, consider using splines

6.3 Newton (Divided Differences)

Description:
Constructs polynomial using divided differences.
When to Use It:

- You need to add points incrementally
- More stable than Vandermonde
- Efficient evaluation

Method:
Constructs divided differences table
Example:

```
X: 1 2 3 4
Y: 1 0 3 4
```

Results:

- Divided differences table
- Newton coefficients $[c_0, c_1, c_2, \ldots]$
- Polynomial: $P(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots$

Advantages:

- More stable than Vandermonde
- Easy to add points
- Efficient evaluation

Disadvantages:

- Polynomial in non-standard form

6.4 Lagrange

Description:
Constructs polynomial as linear combination of Lagrange basis polynomials.
When to Use It:

- You need theoretical form of polynomial
- Mathematical analysis
- Evaluation at few points

Method:

$$P(x) = \sum y_i L_i(x)$$

where

$$L_i(x) = \prod_{j \neq i} \frac{(x - x_j)}{(x_i - x_j)}$$

Example:

```
X: 0 1 2
Y: 1 2 0
```

Results:

- Basis polynomials $L_i(x)$
- Final combination

Advantages:

- Mathematically elegant form
- Does not require solving systems

Disadvantages:

- Expensive to evaluate
- Same Runge problem as Vandermonde

6.5 Splines

Splines divide the interval into segments and use different polynomials in each.

6.5.1 Linear Splines

Description:
Joins points with straight lines.

When to Use It:

- Data with abrupt changes
- Simplicity over smoothness
- Basic visualization

Form:
$S_i(x) = a_i x + b_i$ for $x \in [x_i, x_{i+1}]$

Advantages:

- Very simple
- No oscillations
- Fast to calculate

Disadvantages:

- Not smooth (discontinuity in derivatives)

6.5.2 Quadratic Splines

Description:
Parabolas that join points with continuity in first derivative.

When to Use It:

- You need more smoothness than linear
- You don't need continuous second derivative

Form:
$S_i(x) = a_i x^2 + b_i x + c_i$

Requirements:

- Minimum 3 points

Advantages:

- Smooth (C^1 continuous)
- Fewer oscillations than single polynomial

Disadvantages:

- Discontinuous second derivative

6.5.3 Cubic Splines

Description:
Cubic polynomials with continuity up to second derivative.

When to Use It:

- Maximum smoothness
- Scientific data
- Animation and graphics
- Standard in professional interpolation

Form:
$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$

Boundary Conditions (Natural Splines):

- $S''(x_0) = 0$
- $S''(x_n) = 0$

Advantages:

- Maximum smoothness (C^2 continuous)
- Minimal oscillations
- Industry standard

Disadvantages:

- More complex to calculate

Spline Comparison:

- Linear: Simple, not smooth
- Quadratic: Medium smoothness
- Cubic: Maximum smoothness (recommended)

7 Mathematical Function Syntax

The application uses **SymPy** to process mathematical functions.

7.1 Basic Operators

Operator	**Symbol**	**Example**
Addition	`+`	`x + 3`
Subtraction	`-`	`x - 5`
Multiplication	`*`	`2*x`
Division	`/`	`x/2`
Power	`**`	`x**2` (x^2)
Parentheses	`()`	`(x + 1)*(x - 2)`

7.2 Mathematical Functions

7.2.1 Powers and Roots

```
x**2          # x squared
x**3          # x cubed
x**0.5        # Square root (also: sqrt(x))
x**(1/3)      # Cube root
sqrt(x)       # Square root
```

7.2.2 Exponentials and Logarithms

```
exp(x)        # e^x
log(x)        # Natural logarithm (ln x)
ln(x)         # Natural logarithm (alias of log)
log(x, 10)    # Base 10 logarithm
log(x, 2)     # Base 2 logarithm
```

7.2.3 Trigonometric

```
sin(x)        # Sine
cos(x)        # Cosine
tan(x)        # Tangent
cot(x)        # Cotangent
sec(x)        # Secant
csc(x)        # Cosecant
```

7.2.4 Inverse Trigonometric

```
asin(x)       # Arcsine
acos(x)       # Arccosine
atan(x)       # Arctangent
```

7.2.5 Hyperbolic

```
sinh(x)        # Hyperbolic sine
cosh(x)        # Hyperbolic cosine
tanh(x)        # Hyperbolic tangent
```

7.2.6 Other Functions

```
abs(x)         # Absolute value |x|
sign(x)        # Sign of x (-1, 0, 1)
```

7.3 Constants

```
E              # Number e (2.71828...)
pi             # Number pi (3.14159...)
```

7.4 Valid Function Examples

7.4.1 Polynomials

```
x**3 - 2*x + 1
x**4 - 5*x**3 + 6*x**2 - x + 2
(x + 1)*(x - 2)*(x + 3)
```

7.4.2 Trigonometric

```
sin(x) - 0.5
cos(x)**2 + sin(x)**2 - 1
tan(x) - x
sin(2*x) - cos(x)
```

7.4.3 Exponentials

```
exp(x) - 3
exp(-x**2)
2**x - 8
```

7.4.4 Logarithmic

```
log(x) - 2
ln(x**2 + 1)
log(x, 10) - 1
```

7.4.5 Combined

```
x*exp(-x) - 0.1
sin(x)/x - 0.5
log(sin(x)**2 + 1) - (1/2)
x**2 * cos(x) - 1
exp(x) * sin(x) - 1
```

7.5 Common Errors

Incorrect:

```
2x              # Missing *
x^2             # Use **, not superscript
sen(x)          # Must be sin(x)
^               # Use **, not ^
x**2 + 1)       # Unbalanced parentheses
```

Correct:

```
2*x
x**2
sin(x)
x**2 + 1
```

8 Complete Practical Examples

8.1 Example: Find $\sqrt{2}$

Problem: Find the square root of 2 using Newton-Raphson.
Solution:

1. **Formulate:** $\sqrt{2}$ is root of $f(x) = x^2 - 2$
2. **Parameters:**
   - Function: `x**2 - 2`
   - Derivative: `2*x`
   - x_0: `1.5`
   - Tolerance: `1e-10`
   - Nmax: `20`
3. **Steps in the Calculator:**
   - Select "Equation Solving" tab
   - Click on "Newton-Raphson"
   - Enter function and derivative
   - Click "Calculate"
4. **Expected Result:**
   - Root: 1.414213562373095...
   - Converges in ∼5 iterations

8.2 Example: System of 3 Equations

Problem: Solve the system:

$$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

Solution:

1. **Matrix Format:** $Ax = b$
2. **Parameters:**
   - Matrix A:
     ```
     2 1 -1
     -3 -1 2
     -2 1 2
     ```
   - Vector b: `8 -11 -3`
3. **Method:** Gaussian Elimination with Partial Pivoting
4. **Steps:**
   - "Linear Systems" tab

- Click "Partial Pivot"
- Enter matrix and vector
- Calculate

5. **Result:** $x = [2, 3, -1]$
6. **Verification:**
    - $2(2) + 3 - (-1) = 8$
    - $-3(2) - 3 + 2(-1) = -11$
    - $-2(2) + 3 + 2(-1) = -3$

8.3 Example: Experimental Data Interpolation

Problem: You have temperature vs time measurements:

Time (min)	0	1	2	3	4
Temp (C)	20	25	28	30	31

Construct a function that interpolates this data.
Solution with Cubic Spline:

1. **Parameters:**
   - X values: `0 1 2 3 4`
   - Y values: `20 25 28 30 31`
2. **Method:** Cubic Spline
3. **Result:**
   - 4 cubic polynomials (one for each interval)
   - Smooth function passing through all points
4. **Use:**
   - Estimate temperature at $t = 2.5$ min
   - Evaluate corresponding spline

8.4 Example: Method Comparison

Problem: Find root of $f(x) = x^3 - x - 2$ near $x = 1.5$
Comparison:

Method	**Iterations**	**Precision**	**Comments**
Bisection $[1, 2]$	20	10^{-6}	Slow but safe
False Position	15	10^{-6}	Faster
Newton ($x_0 = 1.5$)	5	10^{-10}	Very fast
Secant ($x_0 = 1, x_1 = 2$)	6	10^{-10}	Fast without derivative

Conclusion:

- For guarantee: Bisection
- For speed with derivative: Newton
- For speed without derivative: Secant

9 Results Interpretation

9.1 Iteration Tables

9.1.1 For Equation Methods

Typical columns:

- **iter**: Iteration number
- x_i: Current approximation
- $f(x_i)$: Function value
- **E**: Absolute error $|x_i - x_{i-1}|$

Interpretation:

- **E decreases**: Method converges
- **E increases**: Method diverges $\times$
- **E stagnant**: Slow convergence or stagnation

Stopping Criterion:

- $E <$ Tolerance: Solution found
- iter = Nmax: Increase Nmax or change method

9.1.2 For Iterative Methods (Systems)

Typical columns:

- **iter**: Iteration
- **E**: Error (norm of $|x^{(n)} - x^{(n-1)}|$)
- $x_0, x_1, x_2, \ldots$: Solution components

Interpretation:

- **Spectral radius** < 1: Method converges
- **Spectral radius** ≥ 1: Method may diverge

9.2 Warning Messages

9.2.1 "Function does not change sign"

- **Cause:** No root in interval $[a, b]$
- **Solution:** Use Incremental Search to find valid interval

9.2.2 "Zero pivot encountered"

- **Cause:** Division by zero in elimination
- **Solution:** Use method with pivoting

9.2.3 "Matrix is not positive definite"

- **Cause:** Cholesky requires positive definite matrix
- **Solution:** Use LU instead of Cholesky

9.2.4 "Spectral radius ≥ 1"

- **Cause:** Iterative method may not converge
- **Solution:**
  - Verify diagonal dominance
  - Adjust omega in SOR
  - Use direct method

9.2.5 "Tolerance is very large"

- **Suggestion:** Result may be imprecise
- **Action:** Reduce tolerance for greater precision

9.3 Graphics

Graph Elements:

- **Function** $f(x)$: Main curve
- **X axis**: x values
- **Y axis**: $f(x)$ values
- **X-axis crossing points**: Roots of $f(x) = 0$

Use:

- Visualize function behavior
- Identify number and location of roots
- Verify results

Desmos Interactivity:

- Zoom: Mouse wheel
- Pan: Drag
- Reset: Click home icon

10 Use Cases

10.1 Engineering

10.1.1 Structural Analysis

- **System:** Equilibrium equations
- **Method:** Gaussian Elimination or LU
- **Example:** Calculate forces in truss

10.1.2 Electrical Circuits

- **System:** Kirchhoff's laws
- **Method:** Iterative methods for large networks
- **Example:** Mesh currents

10.1.3 Heat Transfer

- **Method:** Finite differences $\to$ Linear system
- **Solver:** Gauss-Seidel or SOR
- **Use:** Temperature distribution

10.2 Sciences

10.2.1 Chemistry

- **Problem:** Chemical equilibrium
- **Method:** Newton-Raphson
- **Example:** Equilibrium constants

10.2.2 Physics

- **Problem:** Trajectories, energies
- **Method:** Various depending on problem
- **Example:** Planetary orbits

10.2.3 Biology

- **Problem:** Population models
- **Method:** Interpolation for experimental data
- **Example:** Population growth

10.3 Finance

10.3.1 IRR Calculation (Internal Rate of Return)

- **Equation:** NPV = 0
- **Method:** Bisection or Newton
- **Use:** Evaluate investment projects

10.3.2 Option Models

- **System:** Discretized Black-Scholes equations
- **Method:** Linear systems
- **Use:** Derivative valuation

10.4 Education

10.4.1 Manual Calculation Verification

- **Use:** Check exercises
- **Advantage:** See complete iteration table

10.4.2 Method Comparison

- **Use:** Understand differences
- **Example:** Why is Newton faster?

10.4.3 Visualization

- **Use:** Graphics to understand concepts
- **Example:** See convergence graphically

11 Troubleshooting

11.1 Common Problems

11.1.1 “Application Error”

- **Cause:** Error in function syntax
- **Solution:**
  - Verify syntax (use * for multiplication)
  - Use parentheses correctly
  - Consult syntax section

11.1.2 Incorrect Result

Possible causes:

- Incorrectly entered function
- Incorrect parameters
- Inappropriate method for the problem

Verification:

- Evaluate $f(\text{root}) \approx 0$
- Substitute solution in original system
- Compare with manual calculation

11.1.3 Does Not Converge

For Equations:

- Try better initial x_0
- Increase Nmax
- Change method

For Iterative Systems:

- Verify diagonal dominance
- Adjust omega (SOR)
- Use direct method

11.1.4 Graph Not Displayed

Causes:

- Lost internet connection
- Function with restricted domain
- Syntax error

Solution:

- Verify connection
- Simplify function to test
- Reload page

11.2 Best Practices

11.2.1 Choosing Appropriate Method

For Equations:

1. Do you have interval $[a, b]$? $\to$ Bisection/False Position
2. Can you calculate $f'(x)$? $\to$ Newton
3. Want speed without derivative? $\to$ Secant
4. Explore roots? $\to$ Incremental Search first

For Linear Systems:

1. Small system (< 100)? $\to$ Gaussian/LU
2. Large and sparse system? $\to$ Iterative (Jacobi/GS/SOR)
3. Symmetric positive definite matrix? $\to$ Cholesky
4. Maximum stability? $\to$ Total Pivoting

For Interpolation:

1. Few points (< 10)? $\to$ Polynomial (Newton/Lagrange)
2. Many points? $\to$ Splines
3. Maximum smoothness? $\to$ Cubic Spline
4. Simplicity? $\to$ Linear Spline

11.2.2 Validate Results

Equations:

Verify: $f(\text{root}) \approx 0$

Systems:

Verify: $Ax \approx b$
Calculate: $\|Ax - b\|$

Interpolation:

Verify: $P(x_i) = y_i$ for all points

11.2.3 Precision vs Cost

High Precision (Small Tolerance):

- More iterations
- More time
- Better result

Low Precision (Large Tolerance):

- Fewer iterations

- Faster
- May be sufficient

Recommendations:

- Engineering: 10^{-6} to 10^{-10}
- Finance: 10^{-8}
- Education: 10^{-6}

12 Glossary

Convergence Property that iterations approach the solution.

Linear Convergence Error reduces by constant factor each iteration.

Quadratic Convergence Error reduces to square each iteration (very fast).

Diagonally Dominant Matrix where each diagonal element is greater than sum of others in its row.

Absolute Error |approximate value − previous value|

Relative Error |absolute error/approximate value|

Factorization Decomposition of matrix into product of simpler matrices.

Interpolation Construct function that passes through given points.

Iteration One step of the numerical method.

Positive Definite Matrix Symmetric matrix with all positive eigenvalues.

Singular Matrix Non-invertible matrix (determinant = 0).

Sparse Matrix Matrix with many zeros.

Symmetric Matrix Matrix where $A_{ij} = A_{ji}$.

Triangular Matrix Matrix with zeros above or below diagonal.

Nmax Maximum number of iterations allowed.

Pivot Diagonal element used in elimination.

Pivoting Row/column exchange to improve stability.

Polynomial Function of the form $a_0 + a_1x + a_2x^2 + \cdots$

Spectral Radius Maximum absolute value of eigenvalues (determines convergence).

Root Value x where $f(x) = 0$.

Multiple Root Root where $f(x) = f'(x) = 0$.

Linear System Set of linear equations $Ax = b$.

Spline Piecewise function (different polynomials in each interval).

Tolerance Desired precision (stopping criterion).

13 References

13.1 Recommended Books

1. **Burden, R. L., & Faires, J. D.** (2010). *Numerical Analysis* (9th ed.). Brooks/Cole.

2. **Chapra, S. C., & Canale, R. P.** (2014). *Numerical Methods for Engineers* (7th ed.). McGraw-Hill.

3. **Press, W. H., et al.** (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.

4. **Kincaid, D., & Cheney, W.** (2009). *Numerical Analysis: Mathematics of Scientific Computing* (3rd ed.). American Mathematical Society.

13.2 Online Resources

- **Desmos Calculator**: https://www.desmos.com/calculator
- **SymPy Documentation**: https://docs.sympy.org/
- **NumPy Documentation**: https://numpy.org/doc/

13.3 Mathematical Concepts

Intermediate Value Theorem: If $f(a)$ and $f(b)$ have opposite signs, there exists a root in $[a, b]$.
Banach Fixed Point Theorem: If $|g'(x)| < 1$, then $x = g(x)$ converges.
Sassenfeld Convergence Criterion: For iterative methods.
Diagonal Dominance Condition: Sufficient for convergence of Jacobi and Gauss-Seidel.

14 Appendix: Quick Formulas

Equation Methods

Bisection:

$$x_m = \frac{a+b}{2}$$

False Position:

$$x_m = \frac{f(b) \cdot a - f(a) \cdot b}{f(b) - f(a)}$$

Fixed Point:

$$x_{n+1} = g(x_n)$$

Newton:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Secant:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Newton Multiple Roots:

$$x_{n+1} = x_n - \frac{f(x) \cdot f'(x)}{[f'(x)]^2 - f(x) \cdot f''(x)}$$

Iterative Methods

Jacobi:

$$x^{(n+1)} = T \cdot x^{(n)} + C$$

where $T = -D^{-1}(L+U)$ and $C = D^{-1}b$

Gauss-Seidel:

$$T = -(D-L)^{-1}U, \quad C = (D-L)^{-1}b$$

SOR:

$$T = (D - \omega L)^{-1}[(1-\omega)D + \omega U], \quad C = \omega(D - \omega L)^{-1}b$$

15 Frequently Asked Questions (FAQ)

Q: Does the application save my calculations?
A: No, the application does not save data. Each session is independent.

Q: Can I export results?
A: You can copy and paste the results, or use the browser's print function (Ctrl+P).

Q: Does it work offline?
A: No, it requires connection to load the application and graphics.

Q: Is there a calculation limit?
A: No limit. You can do all the calculations you need.

Q: Can I trust the results for professional work?
A: The methods are correct, but always verify critical results with other tools.

Q: Why doesn't my method converge?
A: Review the "Troubleshooting" section and verify that the parameters are adequate.

Q: How do I report an error?
A: Contact the site administrator with details of the problem and parameters used.

16 Contact and Support

For questions, suggestions or to report problems:

- **Email**: scadavidz@eafit.edu.co
- **Documentation**: This manual

Date: November 2025
Language: English

Thank you for using the Numerical Analysis Calculator!