
Einführung in die Programmierung für Physiker

Die Programmiersprache C – Verwendung wissenschaftlicher Bibliotheken

Marc Wagner

Institut für theoretische Physik
Johann Wolfgang Goethe-Universität Frankfurt am Main

SoSe 2024

GSL (GNU Scientific Library)

- Es existieren zahlreiche frei verfügbare wissenschaftliche Bibliotheken, in denen numerische Standardverfahren (z.B. zur Nullstellensuche, zur Integration, zum Lösen von Differentialgleichungen) implementiert sind.
- Häufig ist die Verwendung solcher wissenschaftliche Bibliotheken zeitsparender und weniger fehleranfällig, als das selbstständige Implementieren eines entsprechenden numerischen Verfahrens.
- Eine oft verwendete wissenschaftliche Bibliothek ist **GSL (GNU Scientific Library)**.

The screenshot shows the GNU Scientific Library (GSL) website. At the top, there is a navigation bar with links for 'About GNU', 'Philosophy', 'Licenses', 'Education', 'Software', 'Documentation', and 'Help GNU'. A search bar is also present. Below the navigation bar, the main heading reads 'GSL - GNU Scientific Library'. The page content includes an 'Introduction' section with the following text:

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License.

The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total with an extensive test suite.

The complete range of subject areas covered by the library includes,

Complex Numbers	Roots of Polynomials
Special Functions	Vectors and Matrices
Permutations	Sorting

- **GSL** enthält ein breites Spektrum von **C**-Funktionen zur Lösung numerischer Standardprobleme.

The screenshot shows the same GNU Scientific Library (GSL) website, but with a more detailed list of subject areas covered by the library. The text reads:

The complete range of subject areas covered by the library includes,

Complex Numbers	Roots of Polynomials
Special Functions	Vectors and Matrices
Permutations	Sorting
BLAS Support	Linear Algebra
Eigensystems	Fast Fourier Transforms
Quadrature	Random Numbers
Quasi-Random Sequences	Random Distributions
Statistics	Histograms
N-Tuples	Monte Carlo Integration
Simulated Annealing	Differential Equations
Interpolation	Numerical Differentiation
Chebyshev Approximation	Series Acceleration
Discrete Hankel Transforms	Root-Finding
Minimization	Least-Squares Fitting
Physical Constants	IEEE Floating-Point
Discrete Wavelet Transforms	Basis splines

Unlike the licenses of proprietary numerical libraries the license of GSL does not restrict scientific cooperation. It allows you to share your programs freely with others.

Downloading GSL

Beispiel: Numerische Integration mit GSL

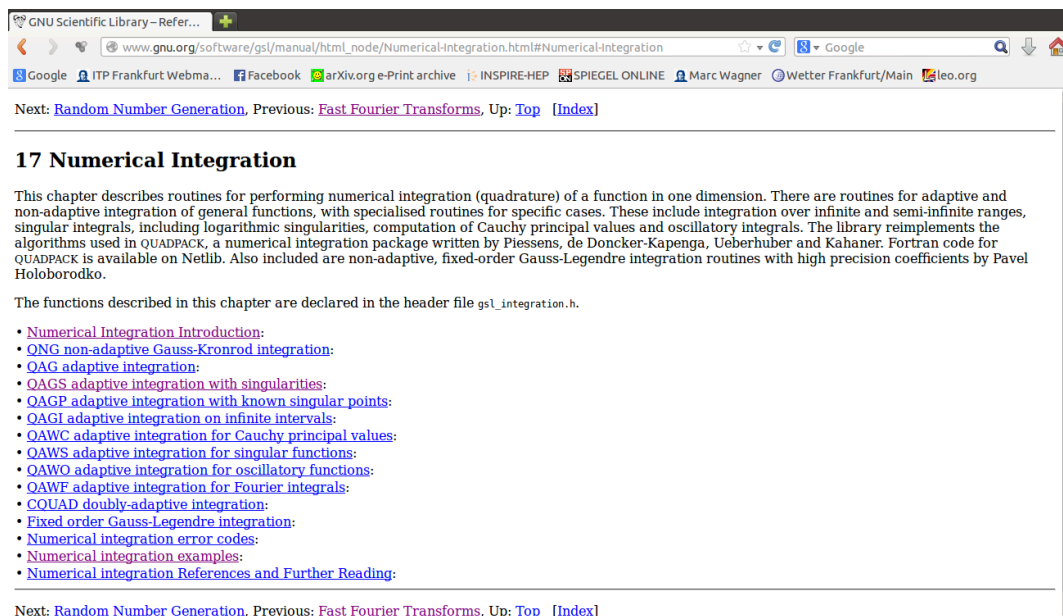
- Zur Illustration sollen die folgenden drei eindimensionalen (auch leicht analytisch lösbaren) Integrale numerisch mit Hilfe von **GSL** gelöst werden.

$$\int_0^1 dx x = \frac{1}{2}$$

$$\int_0^\pi dx (\sin(x))^2 = \frac{\pi}{2}$$

$$\int_0^1 dx \frac{1}{\sqrt{ax}} = \frac{2}{\sqrt{a}} \quad (\text{numerisch im Spezialfall } a = 3)$$

- **GSL** enthält eine Reihe von **C**-Funktionen, in denen unterschiedliche Integrationsverfahren realisiert sind; unter Umständen erfordert die Auswahl des geeignetsten oder zumindest eines erfolgversprechenden Verfahrens für das vorliegende Problem Kenntnisse in numerischer Mathematik (z.B. Inhalt der Vorlesung "Numerische Methoden der Physik", siehe nächste Folie).



The screenshot shows a web browser displaying the GNU Scientific Library (GSL) manual page for Numerical Integration. The page title is "17 Numerical Integration". The content describes routines for performing numerical integration (quadrature) of a function in one dimension. It mentions various routines for adaptive and non-adaptive integration, including integration over infinite and semi-infinite ranges, singular integrals, and Cauchy principal values. The page also lists several specific integration routines and provides a list of links for further reading, including "Numerical Integration Introduction", "QNG non-adaptive Gauss-Kronrod integration", "QAG adaptive integration", "QAGS adaptive integration with singularities", "QAGP adaptive integration with known singular points", "QAGI adaptive integration on infinite intervals", "QAWC adaptive integration for Cauchy principal values", "QAWS adaptive integration for singular functions", "QAWO adaptive integration for oscillatory functions", "QAWF adaptive integration for Fourier integrals", "COUAD doubly-adaptive integration", "Fixed order Gauss-Legendre integration", "Numerical integration error codes", "Numerical integration examples", and "Numerical integration References and Further Reading".

- Die obigen Integrale sind numerisch unproblematisch (endlicher Integrationsbereich, keine starken Oszillationen), lediglich das letzte Integral hat einen singulären Integranden; verwende daher ein Verfahren, das mit solchen Singularitäten zurechtkommt (→ "QAGS adaptive integration with singularities"; auch in "Numerical integration examples" verwendet).

17.4 QAGS adaptive integration with singularities

The presence of an integrable singularity in the integration region causes an adaptive routine to concentrate new subintervals around the singularity. As the subintervals decrease in size the successive approximations to the integral converge in a limiting fashion. This approach to the limit can be accelerated using an extrapolation procedure. The QAGS algorithm combines adaptive bisection with the Wynn epsilon-algorithm to speed up the integration of many types of integrable singularities.

Function: `int gsl_integration_qags (const gsl_function *f, double a, double b, double epsabs, double epsrel, size_t limit, gsl_integration_workspace *workspace, double *result, double *abserr)`

This function applies the Gauss-Kronrod 21-point integration rule adaptively until an estimate of the integral of f over (a,b) is achieved within the desired absolute and relative error limits, $epsabs$ and $epsrel$. The results are extrapolated using the epsilon-algorithm, which accelerates the convergence of the integral in the presence of discontinuities and integrable singularities. The function returns the final approximation from the extrapolation, $result$, and an estimate of the absolute error, $abserr$. The subintervals and their results are stored in the memory provided by $workspace$. The maximum number of subintervals is given by $limit$, which may not exceed the allocated size of the workspace.

```
1. #include<math.h>
2. #include<stdio.h>
3.
4. #include<gsl/gsl_integration.h>
5.
6. // *****
7.
8. // Die Funktionen, die integriert werden.
9.
10. // f(x) = x
11. double f(double x, void *params)
12. {
13.     return x;
14. }
15.
16. // g(x) = sin^2(x)
17. double g(double x, void *params)
18. {
19.     return pow(sin(x), 2.0);
20. }
21.
22. // h(x) = 1/sqrt(a*x)
23. double h(double x, void *params)
24. {
25.     double a = *(double *)params;
26.     return 1.0/sqrt(a*x);
27. }
28.
29. // *****
30.
31. int main(void)
32. {
33.     // int gsl_integration_qags (const gsl_function *f, double a, double b,
34.     // double epsabs, double epsrel, size_t limit,
35.     // gsl_integration_workspace *workspace, double *result, double *abserr)
36.
37.     // Eine GSL-Struktur fuer mathematische Funktionen.
38.     gsl_function func;
39.
40.     // GSL benoetigt einen speziellen Speicherbereich zur Integration; dieser
41.     // kann im vorliegenden Fall bis zu 1000 Intervalle speichern.
```

```

42. gsl_integration_workspace *workspace = gsl_integration_workspace_alloc(1000);
43.
44. double result, abserr;
45. double result_analytically;
46.
47. // *****
48.
49. //  $\int_0^1 dx x = 1/2$ 
50.
51. result_analytically = 0.5;
52.
53. func.function = &f;
54. func.params = NULL;
55.
56. // Die eigentliche numerische Integration.
57. gsl_integration_qags(&func, 0.0, 1.0, 0.0, 1.0e-7, 1000, workspace, &result, &abserr);
58.
59. printf("int_0^1 dx x = ...\n");
60. printf(" ... = %.12lf (numerically)\n", result);
61. printf(" ... = %.12lf (analytically)\n", result_analytically);
62. printf(" estimated error = % .12f\n", abserr);
63. printf(" actual error   = % .12f\n", fabs(result - result_analytically));
64. printf(" intervals = %zd\n", workspace->size);
65.
66. // *****
67. printf("\n");
68. // *****
69.
70. //  $\int_0^{\pi} dx (\sin(x))^2 = \pi/2$ 
71.
72. result_analytically = 0.5*M_PI;
73.
74. func.function = &g;
75. func.params = NULL;
76.
77. // Die eigentliche numerische Integration.
78. gsl_integration_qags(&func, 0.0, M_PI, 0.0, 1.0e-7, 1000, workspace, &result, &abserr);
79.
80. printf("int_0^{\pi} dx (\sin(x))^2 = ...\n");
81. printf(" ... = %.12lf (numerically)\n", result);
82. printf(" ... = %.12lf (analytically)\n", result_analytically);
83. printf(" estimated error = % .12f\n", abserr);
84. printf(" actual error   = % .12f\n", fabs(result - result_analytically));
85. printf(" intervals = %zd\n", workspace->size);
86.
87. // *****
88. printf("\n");
89. // *****
90.
91. //  $\int_0^1 dx 1/\sqrt{a*x} = 2*\sqrt{a*x}/a|_0^1 = 2/\sqrt{a}$ 
92.
93. double a = 3.0;
94. result_analytically = 2.0/sqrt(a);
95.
96. func.function = &h;
97. func.params = &a;
98.
99. // Die eigentliche numerische Integration.
100. gsl_integration_qags(&func, 0.0, 1.0, 0.0, 1.0e-7, 1000, workspace, &result, &abserr);

```

```

101.
102. printf("int_0^1 dx 1/sqrt(3*x) = ...\n");
103. printf(" ... = +%.12lf (numerically)\n", result);
104. printf(" ... = +%.12lf (analytically)\n", result_analytically);
105. printf(" estimated error = % .12f\n", abserr);
106. printf(" actual error   = % .12f\n", fabs(result - result_analytically));
107. printf(" intervals = %zd\n", workspace->size);
108.
109. // *****
110.
111. gsl_integration_workspace_free(workspace);
112. }

```

- Beim Kompilieren muss die **GSL**-Bibliothek eingebunden werden; unter **Linux** und bei Verwendung der Compiler **gcc** oder **g++** dient dafür die Option **-l`libname`**:
 - **-lgsl**: **GSL** wird eingebunden.
 - **-lgslcbblas**: lineare Algebra für **GSL** wird eingebunden (BLAS = Basic Linear Algebra Subprograms).

```

mwagner@laptop-tigger:~/lecture_ProgPhys/slides/tmp$ ls -l
insgesamt 4
-rw-rw-r-- 1 mwagner mwagner 2988 Jan 21 16:00 prog.c
mwagner@laptop-tigger:~/lecture_ProgPhys/slides/tmp$ g++ -o 09_integratation_with_gsl 09_integratation_with_gsl.c -lgsl -lgslcbblas
mwagner@laptop-tigger:~/lecture_ProgPhys/slides/tmp$ ls -l
insgesamt 20
-rwxrwxr-x 1 mwagner mwagner 13555 Jan 21 17:13 prog
-rw-rw-r-- 1 mwagner mwagner 2988 Jan 21 16:00 prog.c
mwagner@laptop-tigger:~/lecture_ProgPhys/slides/tmp$ ./09_integratation_with_gsl
int_0^1 dx x = ...
... = +0.500000000000 (numerically)
... = +0.500000000000 (analytically)
estimated error = 0.000000000000
actual error   = 0.000000000000
intervals = 1

int_0^pi dx (sin(x))^2 = ...
... = +1.570796326795 (numerically)
... = +1.570796326795 (analytically)
estimated error = 0.000000000000
actual error   = 0.000000000000
intervals = 1

int_0^1 dx 1/sqrt(3*x) = ...
... = +1.154700538379 (numerically)
... = +1.154700538379 (analytically)
estimated error = 0.000000000000
actual error   = 0.000000000000
intervals = 6

```

"Numerische Methoden der Physik" (WS 2023/24)

Inhalt

- **1 Introduction**
- **2 Representation of numbers in computers, roundoff errors**
 - 2.1 Integers
 - 2.2 Real numbers, floating point numbers
 - 2.3 Roundoff errors
 - 2.3.1 Simple examples
 - 2.3.2 Another example: numerical derivative via finite difference
- **3 Ordinary differential equations, initial value problems**
 - 3.1 Physics motivation
 - 3.2 Euler's method
 - 3.3 Runge-Kutta method
 - 3.3.1 Estimation of errors
 - 3.3.2 Adaptive step size
- **4 Dimensionful quantities on a computer**
 - 4.1 Method 1: define units for your computation
 - 4.2 Method 2: use exclusively dimensionless quantities
- **5 Root finding, solving systems of non-linear equations**
 - 5.1 Physics motivation
 - 5.2 Bisection (only for $N=1$)
 - 5.3 Secant method (only for $N=1$)
 - 5.4 Newton-Raphson method (for $N=1$)
 - 5.5 Newton-Raphson method (for $N>1$)
- **6 Ordinary differential equations, boundary value problems**
 - 6.1 Physics motivation
 - 6.2 Shooting method
 - 6.2.1 Example: QM, 1 dimension, infinite potential well
 - 6.2.2 Example: QM, 1 dimension, harmonic oscillator
 - 6.2.3 Example: QM, 3 dimensions, spherically symmetric potential
 - 6.3 Relaxation methods
- **7 Solving systems of linear equations**
 - 7.1 Problem definition, general remarks
 - 7.2 Gauss-Jordan elimination (a direct method)

- 7.2.1 Pivoting
- 7.3 Gauss elimination with back substitution (a direct method)
- 7.4 LU decomposition (a direct method)
 - 7.4.1 Crout's algorithm
 - 7.4.2 Computation of the solution of $Ax = b$
 - 7.4.3 Computation of $\det(A)$
- 7.5 QR decomposition (a direct method)
- 7.6 Iterative refinement of the solution of $Ax = b$ (for direct methods)
- 7.7 Conjugate gradient method (an iterative method)
 - 7.7.1 Symmetric positive definite A
 - 7.7.2 Example: static electric charge inside a grounded box in 2 dimensions
 - 7.7.3 Generalizations
 - 7.7.4 Condition number, preconditioning
- **8 Numerical integration**
 - 8.1 Numerical integration in 1 dimension
 - 8.1.1 Newton-Cotes formulas
 - 8.1.2 Gaussian integration
 - 8.2 Numerical integration in $D \geq 2$ dimensions
 - 8.2.1 Nested 1-dimensional integration
 - 8.2.2 Monte Carlo integration
 - 8.2.3 When to use which method?
- **9 Eigenvalues and eigenvectors**
 - 9.1 Problem definition, general remarks
 - 9.2 Basic principle of numerical methods for eigenvalue problems
 - 9.3 Jacobi method
 - 9.4 Example: molecule oscillations inside a crystal
- **10 Interpolation, extrapolation, approximation**
 - 10.1 Polynomial interpolation
 - 10.2 Cubic spline interpolation
 - 10.3 Method of least squares
 - 10.4 χ^2 minimizing fits
- **11 Function minimization, optimization**
 - 11.1 Problem definition, general remarks
 - 11.2 Golden section search in $D=1$ dimension
 - 11.3 Function minimization using quadratic interpolation in $D=1$ dimension

- 11.4 Function minimization using derivatives in $D=1$ dimension
- 11.5 Function minimization in $D \geq 2$ dimensions by repeated minimization in 1 dimension
- 11.6 Simplex algorithm ($D \geq 2$ dimensions)
- 11.7 Simulated annealing
- **12 Monte Carlo simulation of partition functions**
 - 12.1 Ising model
 - 12.2 Basic principle of Monte Carlo simulations
 - 12.3 Examples of common Monte Carlo algorithms
 - 12.4 Monte Carlo simulation of the Ising model
- **13 Partial differential equations**
 - 13.1 Introduction
 - 13.2 Initial value problems
 - 13.3 Boundary value problems

Literatur

- Numerical Recipes: The Art of Scientific Computing (W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press).