
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

WS 2019/20

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 'Introduction' section states: '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,' followed by a list of topics: Complex Numbers, Special Functions, Permutations, Roots of Polynomials, Vectors and Matrices, and Sorting. A sidebar on the right contains a 'Join the FSF!' sign-up form and a fundraising goal of \$450K.

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

This screenshot shows a more detailed view of the GSL website's subject areas. The text reads: 'The complete range of subject areas covered by the library includes,' followed by a two-column list of topics: Complex Numbers, Special Functions, Permutations, BLAS Support, Eigensystems, Quadrature, Quasi-Random Sequences, Statistics, N-Tuples, Simulated Annealing, Interpolation, Chebyshev Approximation, Discrete Hankel Transforms, Minimization, Physical Constants, Discrete Wavelet Transforms, Roots of Polynomials, Vectors and Matrices, Sorting, Linear Algebra, Fast Fourier Transforms, Random Numbers, Random Distributions, Histograms, Monte Carlo Integration, Differential Equations, Numerical Differentiation, Series Acceleration, Root-Finding, Least-Squares Fitting, IEEE Floating-Point, and Basis splines. Below the list, it states: '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.' The 'Downloading GSL' section is partially visible at the bottom.

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).

Next: [Random Number Generation](#), Previous: [Fast Fourier Transforms](#), Up: [Top](#) [\[Index\]](#)

17 Numerical Integration

This chapter describes routines for performing numerical integration (quadrature) of a function in one dimension. There are routines for adaptive and non-adaptive integration of general functions, with specialised routines for specific cases. These include integration over infinite and semi-infinite ranges, singular integrals, including logarithmic singularities, computation of Cauchy principal values and oscillatory integrals. The library reimplements the algorithms used in QUADPACK, a numerical integration package written by Piessens, de Doncker-Kapenga, Ueberhuber and Kahaner. Fortran code for QUADPACK is available on Netlib. Also included are non-adaptive, fixed-order Gauss-Legendre integration routines with high precision coefficients by Pavel Holoborodko.

The functions described in this chapter are declared in the header file `gsl_integration.h`.

- [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](#):
- [Numerical integration References and Further Reading](#):

Next: [Random Number Generation](#), Previous: [Fast Fourier Transforms](#), Up: [Top](#) [\[Index\]](#)

- 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).

Next: [QAGP adaptive integration with known singular points](#), Previous: [QAG adaptive integration](#), Up: [Numerical Integration](#) [\[Index\]](#)

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 **-llibname**:
 - **-lgsl**: **GSL** wird eingebunden.
 - **-lgslcblas**: 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 prog prog.c -lgsl -lgslcblas
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$ ./prog
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

```

Inhalt

- **1 Einleitung**
- **2 Darstellung von Zahlen, Rundungsfehler**
 - 2.1 Ganze Zahlen (Integer)
 - 2.2 Gleitkommazahlen (= reelle Zahlen)
 - 2.3 Rundungsfehler
 - 2.3.1 Elementare Beispiele
 - 2.3.2 Numerische Ableitung
- **3 Gewöhnliche Differentialgleichungen, Anfangswertprobleme**
 - 3.1 Physikalische Motivation
 - 3.2 Euler-Methode
 - 3.3 Runge-Kutta-Methode
 - 3.3.1 Fehlerabschätzung
 - 3.3.2 Dynamische Anpassung der Schrittweite
- Einschub: Einheitenbehaftete/dimensionslose Größen
- **4 Nullstellensuche, lösen nicht-linearer Gleichungen**
 - 4.1 Problemstellung, physikalische Motivation
 - 4.2 Bisektion (1 Gleichung, 1 Variable)
 - 4.3 Sekantenverfahren (1 Gleichung, 1 Variable)
 - 4.4 Newton-Raphson-Verfahren (1 Gleichung, 1 Variable)
 - 4.5 Newton-Raphson-Verfahren ($N > 1$ Gleichungen, N Variablen)
- **5 Gewöhnliche Differentialgleichungen, Randwertprobleme**
 - 5.1 Problemstellung, physikalische Motivation
 - 5.2 Shooting-Methode
 - 5.2.1 Beispiel: Quantenmechanik, eindimensionaler unendlicher Potentialtopf
 - 5.2.2 Beispiel: Quantenmechanik, eindimensionaler harmonischer Oszillator
 - 5.2.3 Beispiel: Quantenmechanik, dreidimensionale radialsymmetrische Probleme
 - 5.3 Relaxation-Methoden
- **6 Lösen linearer Gleichungssysteme**
 - 6.1 Problemstellung
 - 6.2 Gauß-Jordan-Elimination
 - 6.2.1 Pivotisierung
 - 6.3 Gauß-Elimination mit Rückwärtssubstitution
 - 6.4 LU-Zerlegung
 - 6.4.1 Crouts Algorithmus
 - 6.4.2 Lösen von $Ax = b$ mittels LU-Zerlegung
 - 6.4.3 Berechnen von $\det(A)$ mittels LU-Zerlegung
 - 6.5 QR-Zerlegung
 - 6.6 Iterative Verbesserung einer Lösung
 - 6.7 Methode der konjugierten Gradienten
 - 6.7.1 Spezialfall: Matrix symmetrisch und positiv definit

- 6.7.2 Verallgemeinerungen
- 6.7.3 Konditionszahl einer Matrix, Vorkonditionierung
- **7 Numerische Integration**
 - 7.1 Eindimensionale Integration
 - 7.1.1 Newton-Cotes-Formeln
 - 7.1.2 Gaußsche Integralformeln
 - 7.2 Mehrdimensionale Integration
 - 7.2.1 Geschachtelte eindimensionale Integration
 - 7.2.2 Monte-Carlo-Integration
 - 7.2.3 Wann ist welches Verfahren geeignet?
- **8 Eigenwertprobleme**
 - 8.1 Problemstellung, grundlegende Eigenschaften und Tatsachen
 - 8.2 Prinzipielle Arbeitsweise numerischer Eigenwertverfahren
 - 8.3 Jacobi-Verfahren
 - 8.4 Beispiel für physikalische Anwendung: Kleine Schwingungen
 - 8.5 Bibliotheken für Eigenwertprobleme
- Einschub: Verwendung numerischer Bibliotheken
- **9 Interpolation, Extrapolation, Approximation**
 - 9.1 Polynominterpolation
 - 9.2 Kubische Spline-Interpolation
 - 9.3 Methode der kleinsten Fehlerquadrate
 - 9.4 χ^2 -Fitting
- **10 Funktionsminimierung, Optimierung**
 - 10.1 "Golden-Section-Search" in $D = 1$ Dimensionen
 - 10.2 Quadratische Interpolation in $D = 1$
 - 10.3 Minimumsuche mit Hilfe von Ableitungen in $D = 1$
 - 10.4 Simplex-Methode ($D > 1$)
 - 10.5 $D > 1$ -Minimierung durch wiederholte $D = 1$ -Minimierung
 - 10.6 Simulated-Annealing
 - 10.6.1 Kombinatorische Minimierung
 - 10.6.2 Kontinuierliche Minimierung
- **11 Monte Carlo-Simulation statistischer Zustandssummen**
 - 11.1 Ising-Modell
 - 11.2 Grundlagen der Monte Carlo-Simulation
 - 11.2.1 Metropolis-Algorithmus
 - 11.2.2 Heatbath-Algorithmus
 - 11.3 Monte Carlo-Simulation des Ising-Modells

Literatur

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