# Einführung in die Programmierung für Physiker

# Die Programmiersprache C – Verwendung wissenschaftlicher Bibliotheken

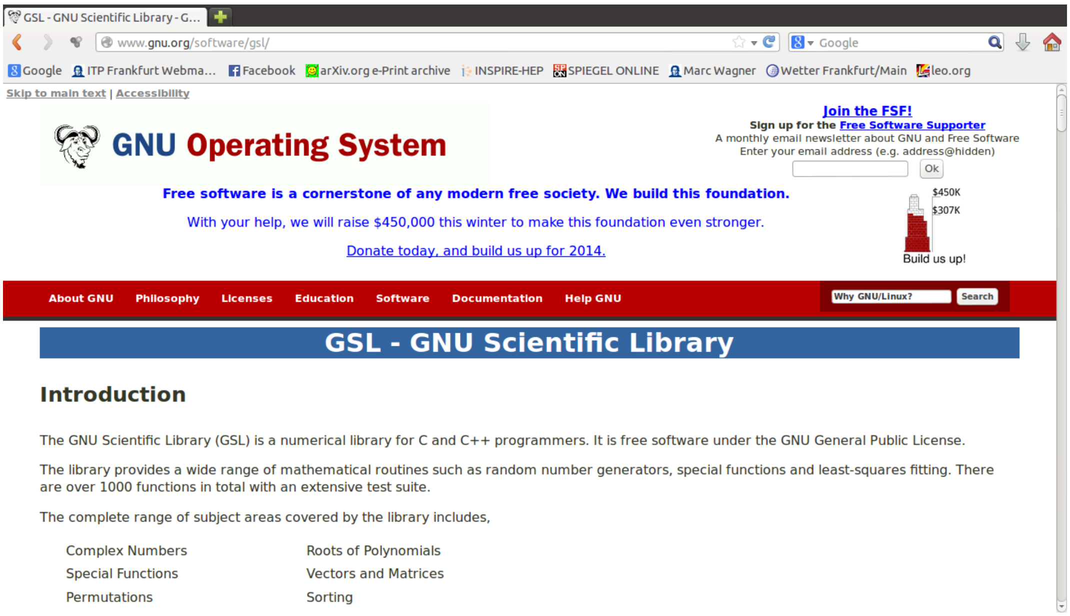
Marc Wagner

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

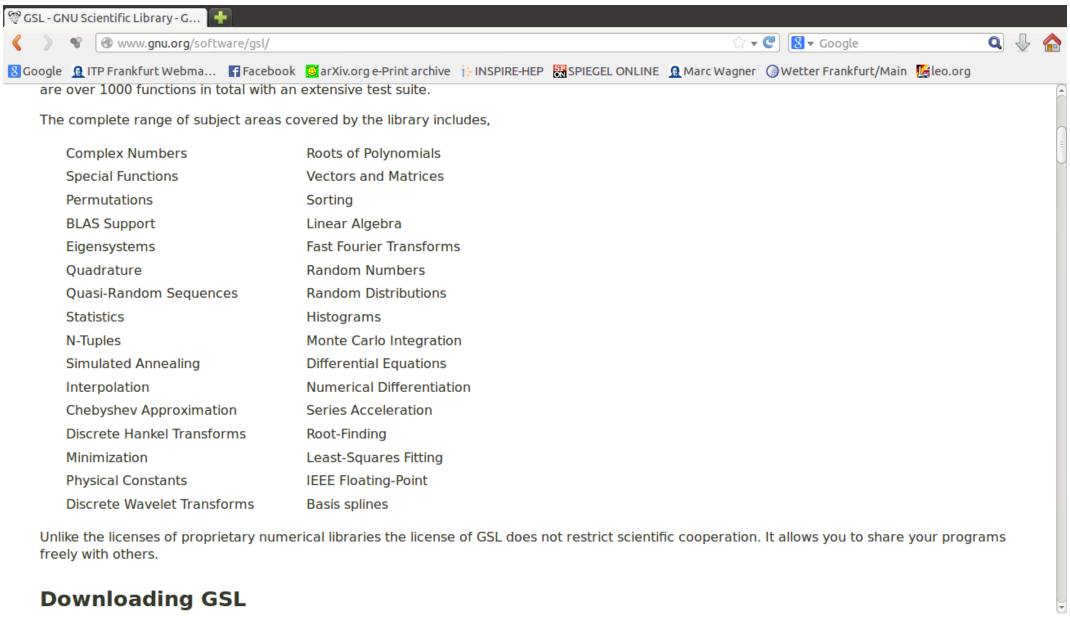
SoSe 2024

# 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)**.




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



# Beispiel: Numerische Integration mit GSL

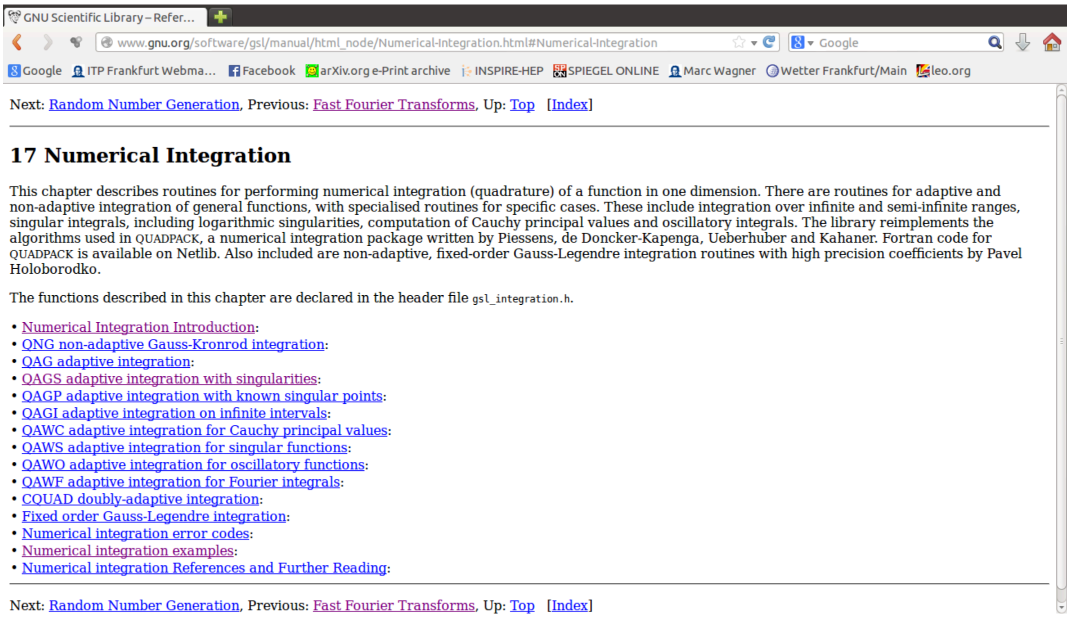
- Zur Illustration sollen die folgenden drei eindimensionalen (auch leicht analytisch lösbaren) Integrale numerisch mit Hilfe von **GSL** gelöst werden.

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

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

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

- **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).



GNU Scientific Library – Refer...

www.gnu.org/software/gsl/manual/html_node/Numerical-Integration.html#Numerical-Integration

Google | ITP Frankfurt Webma... | Facebook | arXiv.org e-Print archive | INSPIRE-HEP | SPIEGEL ONLINE | Marc Wagner | Wetter Frankfurt/Main | leo.org

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:
- CQUAD 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.

```c
#include<math.h>
#include<stdio.h>

#include<gsl/gsl_integration.h>

// ********************

// Die Funktionen, die integriert werden.

// f(x) = x
double f(double x, void *params)
{
  return x;
}

// g(x) = sin^2(x)
double g(double x, void *params)
{
  return pow(sin(x), 2.0);
}

// h(x) = 1/sqrt(a*x)
double h(double x, void *params)
{
  double a = *(double *)params;
  return 1.0/sqrt(a*x);
}

// ********************

int main(void)
{
  // 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)

  // Eine GSL-Struktur fuer mathematische Funktionen.
  gsl_function func;

  // GSL benoetigt einen speziellen Speicherbereich zur Integration; dieser
  // kann im vorliegenden Fall bis zu 1000 Intervalle speichern.
```

```
  gsl_integration_workspace *workspace = gsl_integration_workspace_alloc(1000);

  double result, abserr;
  double result_analytically;

  // **********

  // int_0^1 dx x = 1/2

  result_analytically = 0.5;

  func.function = &f;
  func.params = NULL;

  // Die eigentliche numerische Integration.
  gsl_integration_qags(&func, 0.0, 1.0, 0.0, 1.0e-7, 1000, workspace, &result, &abserr);

  printf("int_0^1 dx x = ...\n");
  printf("  ... = +%.12lf (numerically)\n", result);
  printf("  ... = +%.12lf (analytically)\n", result_analytically);
  printf("  estimated error = % .12f\n", abserr);
  printf("  actual error    = % .12f\n", fabs(result - result_analytically));
  printf("  intervals =  %zd\n", workspace->size);

  // **********
  printf("\n");
  // **********

  // int_0^\pi dx (sin(x))^2 = \pi/2

  result_analytically = 0.5*M_PI;

  func.function = &g;
  func.params = NULL;

  // Die eigentliche numerische Integration.
  gsl_integration_qags(&func, 0.0, M_PI, 0.0, 1.0e-7, 1000, workspace, &result, &abserr);

  printf("int_0^\\pi dx (sin(x))^2 = ...\n");
  printf("  ... = +%.12lf (numerically)\n", result);
  printf("  ... = +%.12lf (analytically)\n", result_analytically);
  printf("  estimated error = % .12f\n", abserr);
  printf("  actual error    = % .12f\n", fabs(result - result_analytically));
  printf("  intervals =  %zd\n", workspace->size);

  // **********
  printf("\n");
  // **********

 // int_0^1 dx 1/sqrt(a*x) = 2*sqrt(a*x)/a|_0^1 = 2/sqrt(a)

  double a = 3.0;
  result_analytically = 2.0/sqrt(a);

  func.function = &h;
  func.params = &a;

  // Die eigentliche numerische Integration.
  gsl_integration_qags(&func, 0.0, 1.0, 0.0, 1.0e-7, 1000, workspace, &result, &abserr);
```

```
    printf("int_0^1 dx 1/sqrt(3*x) = ...\n");
    printf("  ... = +%.12lf (numerically)\n", result);
    printf("  ... = +%.12lf (analytically)\n", result_analytically);
    printf("  estimated error = % .12f\n", abserr);
    printf("  actual error    = % .12f\n", fabs(result - result_analytically));
    printf("  intervals =  %zd\n", workspace->size);

    // **********

    gsl_integration_workspace_free(workspace);
}
```

- 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.
  - **`-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 09_integration_with_gsl 09_integration_with_gsl.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$ ./09_integration_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

    - 11.4 Function minimization using derivatives in D=1 dimension
    - 11.5 Function minimization in D>=2 dimensions by repeated minimization in 1 dimension
    - 11.6 Simplex algorithm (D>=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).