NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 10
Numerical Quadrature

P. Bansal

Problem 10.1: Numerical integration of improper integrals

In this problem, we study two methods for computing improper integrals of the form $\int_{-\infty}^{\infty} f(t)dt$, where $f : \mathbb{R} \to \mathbb{R}$ is a continuous function that decays sufficiently fast for $|t| \to \infty$ (so that it is integrable on $\mathbb{R}$).

One option is to approximate the integral by truncating the domain to a bounded interval $[-b, b],\ b \leq \infty$:

$$I = \int_{-\infty}^{\infty} f(t)dt \approx \int_{-b}^{b} f(t)dt.$$

Then, standard quadrature rules can be used to compute this integral over $[-b, b]$.

(a) Given the integrand $g(t) := 1/(1+t^2)$, find b such that the truncation error E_T satisfies:

$$E_T := \left| \int_{-\infty}^{\infty} g(t)dt - \int_{-b}^{b} g(t)dt \right| \leq \epsilon \tag{10.1}$$

for some $\epsilon > 0$.

SOLUTION:

An antiderivative of g is atan. The function g is even.

$$E_T = 2\int_{b}^{\infty} g(t)dt = 2(\lim_{x\to\infty} \operatorname{atan}(x) - \operatorname{atan}(b)) = \pi - 2\cdot\operatorname{atan}(b) \overset{!}{<} \epsilon \tag{10.2}$$

$$\implies b > \tan((\pi-\epsilon)/2) = \cot(\epsilon/2) \approx 2/\epsilon.$$

(b) What is the challenge in using the truncation approach for a generic integrand?

SOLUTION:

A good choice of b requires a detailed knowledge about the decay of f, which may not be available for f defined implicitly.

Another option is to transform the improper integral I to a bounded domain by substitution. For instance, we may substitute $t = \cot(s)$.

(c) Rewrite the integral I using the substitution $t = \cot(s)$.

SOLUTION:

We have:

$$\frac{dt}{ds} = -\left(1+\cot^2(s)\right) = -\left(1+t^2\right) \tag{10.3}$$

$$\int_{-\infty}^{\infty} f(t)\,dt = -\int_{\pi}^{0} f(\cot(s))\left(1+\cot^2(s)\right)ds = \int_{0}^{\pi} \frac{f(\cot(s))}{\sin^2(s)}\,ds, \tag{10.4}$$

because $\sin^2(\theta) = \frac{1}{1+\cot^2(\theta)}$.

(d) Simplify the transformed integral from subproblem 10.1.(c) explicitly for $g(t) := \frac{1}{1+t^2}$.

Hint: Refer Pythagorean trigonometric identities.

SOLUTION:

Replacing the function $g(t)$, we get:

$$\int_0^{\pi} \frac{1}{1+\cot^2(s)} \frac{1}{\sin^2(s)} ds = \int_0^{\pi} \frac{1}{\sin^2(s)+\cos^2(s)} ds = \int_0^{\pi} ds = \pi$$

(e) Implement a C++ function that uses the substitution $t = \cot(s)$ and the n-point Gauss-Legendre quadrature rule to evaluate $\int_{-\infty}^{\infty} f(t)dt$:

```
template <typename Function>
double quadinf(int n, const Function & f);
```

Use the function `golubwelsh` (see `golubwelsh.hpp`) to compute the nodes and weights of the Gauss quadrature rule.

SOLUTION:

C++-code 10.1: Implementation `quadinf`

```
template <class Function>
double quad(const Function& f,
            const Eigen::VectorXd& w, const Eigen::VectorXd& x,
            double a, double b) {
    double I = 0;
    for(int i = 0; i < w.size(); ++i) {
        I += f( (x(i) + 1) * (b - a) / 2 + a ) * w(i);
    }
    return I * (b - a) / 2.;
}
```

```
template <class Function>
double quadinf(const int n, Function&& f) {
    Eigen::VectorXd w, x;

    // Compute nodes and weights of Gauss quadrature rule
    // using Golub-Welsh algorithm
    golubwelsh(n, w, x);

    //!  NOTE: no function cot available in c++, need to resort to trigonometric identities
    //!  Both lines below are valid, the first computes three trigonometric functions
    auto ftilde = [&f] (double x) { return f(std::cos(x)/std::sin(x)) / pow(std::sin(x),2); };
    /* auto ftilde = [&f] (double x) { double cot = std::tan(PI_HALF - x); return f(cot) * (1. + pow(cot,2)); }; */

```

```
    return quad(ftilde, w, x, 0, PI);
}
```

(f) Study the convergence behaviour of the quadrature method, for $n \to \infty$, implemented in the previous subproblem for the integrand $h(t) := \exp\left(-(t-1)^2\right)$ (shifted Gaussian). For error computation, you can use that $\int_{-\infty}^{\infty} h(t)dt = \sqrt{\pi}$.

SOLUTION:

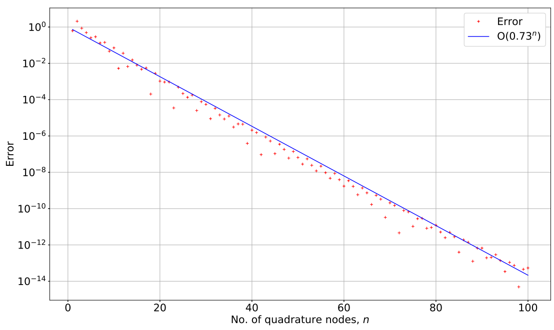


Fig. 1

We observe exponential convergence: The graph of the quadrature error vs. the number of quadrature nodes in lin-log scale is approximately a line with negative slope.

Problem 10.2: Smooth integrand by transformation

Knowledge about the structure of a non-smooth integrand can be used to restore its smoothness by transformation. We explore this idea in the current problem.

Let $f : [-1, 1] \to \mathbb{R}$ be a smooth, odd function. We want to approximate the integral:

$$I := \int_{-1}^{1} \arcsin(t)\, f(t)\, \mathrm{d}t \tag{10.5}$$

using a suitable quadrature rule. Note that $d(\arcsin(t))/dt = 1/\sqrt{1-t^2}$, which is not defined at $t = \pm 1$.

We use Gauss quadrature, however, its detailed knowledge is **not** important to solve this problem. A C++ function `gaussquad` (see `gaussquad.hpp`) to compute the nodes $\mathbf{x}$ and the weights $\mathbf{w}$ of an n-point Gaussian quadrature on $[-1, 1]$ is provided:

```
QuadRule gaussquad(const unsigned n);
```

Here, `QuadRule` is a `struct` containing the quadrature weights `w` and nodes `x`:

```
struct QuadRule {
  Eigen::VectorXd nodes, weights;
};
```

(a) Implement a C++ function:

```
template <class Function>
void nonSmoothIntegrand(const Function & f);
```

to study the convergence behaviour of the quadrature error versus the number of quadrature points $n = 1, \ldots, 50$. Here `f` is a handle to the function f.

SOLUTION:

In our implementation we defined an auxiliary function that computes the integral for a given function and quadrature rule.

C++-code 10.2: gaussConv.cpp

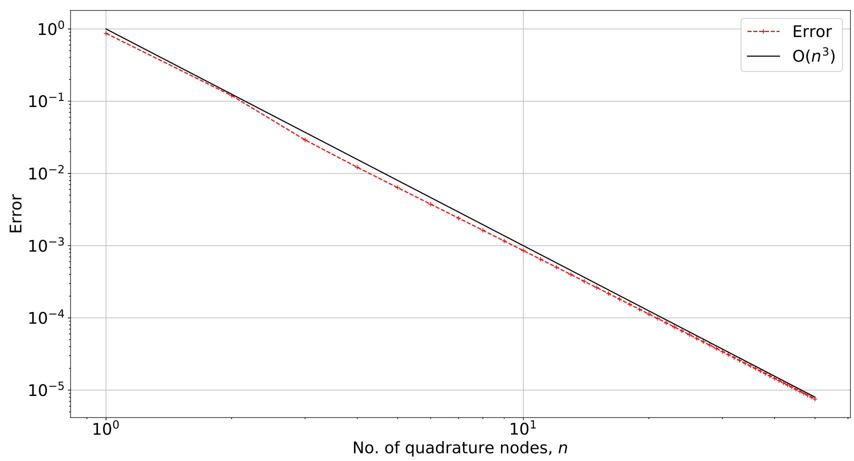
```
template <class Function>
double integrate(const QuadRule& qr, const Function& f) {
    double I = 0;
    for (unsigned i = 0; i < qr.weights.size(); ++i) {
        I += qr.weights(i) * f(qr.nodes(i));
    }
    return I;
}
```

```
template <class Function>
void nonSmoothIntegrand(const Function& fh, const double I_ex) {

    std::vector<double> evals, // used to save no. of quad nodes
              error; // used to save the error
    // Build integrand
```

```cpp
    auto f = [fh](double x) {
        return std::asin(x)*fh(x);
    };

    const unsigned N = 50; // Max. no. of nodes

    for (unsigned n = 1; n <= N; ++n) {
        QuadRule qr;
        gaussquad(n, qr); // Create quadrature rule

        double I = integrate(qr, f); // Compute integral

        evals.push_back(n); // Save no. of quadrature nodes
        error.push_back(std::abs(I - I_ex)); // Save error
    }

    Eigen::Map<Eigen::VectorXd> numNodes(evals.data(), evals.size());
    Eigen::Map<Eigen::VectorXd> quadErr(error.data(), error.size());

    for (int i=0; i<evals.size(); i++)
            std::cout << numNodes(i) << "\t" << quadErr(i) << std::endl;

}
```

(b) Plot the convergence for $f(t) = \sinh(t)$ and describe it qualitatively and quantitatively. Use $I \approx 0.870267525725852642$ as the exact value of the integral.

SOLUTION:



The plot indicates algebraic convergence (being a straight line in log-log mode), of order approximately $O(n^{-3})$.

(c) Transform the integral I using a suitable variable substitution, so that the integrand becomes smooth, if the given $f \in C^\infty([-1,1])$ is smooth.

SOLUTION:

With the change of variable $t = \sin(x)$, $\mathrm{d}t = \cos x \mathrm{d}x$ we obtain

$$I = \int_{-1}^{1} \arcsin(t)\ f(t)\,\mathrm{d}t = \int_{-\pi/2}^{\pi/2} x\ f(\sin(x))\cos(x)\,\mathrm{d}x. \tag{10.6}$$

(The change of variable has to provide a smooth integrand on the integration interval.) The integrand is smooth, provided f is smooth, whence exponential convergence follows.

(d) Implement a C++ function:

```
template <class Function>
void smoothIntegrand(const Function & f);
```

to study the convergence behaviour of the quadrature error versus the number of quadrature points $n = 1, \ldots, 50$, as in Sub-problem (a).

SOLUTION:

The Gauss quadrature nodes and weights are defined on the interval $[-1, 1]$. We need to transform them to the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Consider the following general affine transformation:

$$\begin{cases} \Phi : [-1,1] & \to [a,b] \\ \qquad \hat{x} & \to a + \frac{b-a}{2}(\hat{x}+1) = \frac{b-a}{2}\hat{x} + \frac{a+b}{2} \end{cases} \tag{10.7}$$

Then we can transform $\int_a^b g(x)\,dx$, the integral which needs to be computed over the domain $[a, b]$, as the following:

$$\int_a^b g(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} g\left(\frac{b-a}{2}\hat{x} + \frac{a+b}{2}\right) d\hat{x} \approx \frac{b-a}{2}\sum_{i=1}^{n} \hat{w}_i f\left(\frac{b-a}{2}\hat{x} + \frac{a+b}{2}\right) \tag{10.8}$$

The nodes on $[a, b]$, $\{x_i\}_{i=1}^n$, can be directly obtained from the affine transformation above.

The weights on $[a, b]$, $\{w_i\}_{i=1}^n$, can be obtained by:

$$w_i = \frac{b-a}{2}\hat{w}_i \tag{10.9}$$

In our example, this means that $w_i = \frac{\pi}{2}\hat{w}_i$.

C++-code 10.3: `smoothIntegrand` solution.

```
template <class Function>
void smoothIntegrand(const Function& f, const double I_ex) {
    std::vector<double> evals, // Used to save no.  of quad nodes
                        error; // Used to save the error
    // Transform integrand
```

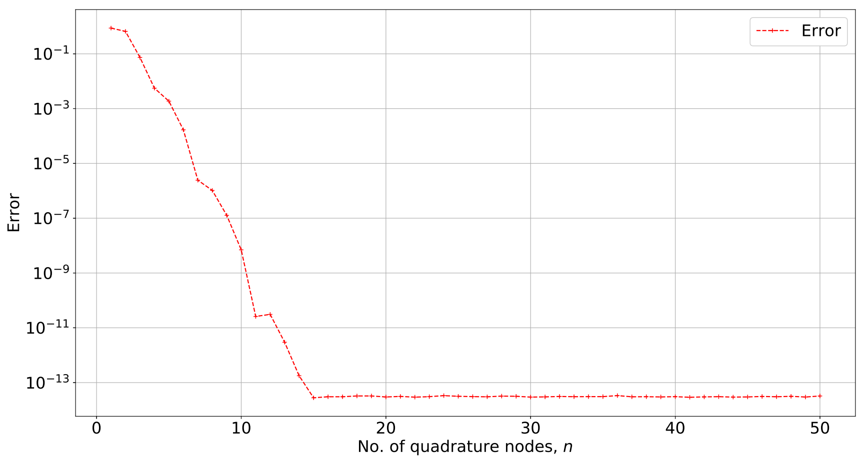
```
    auto g = [f] (double x) {
        return x*f(std::sin(x))*std::cos(x);
    };

    const unsigned N = 50; // Max. no. of nodes

    for (unsigned n = 1; n <= N; ++n) {
        QuadRule qr;
        gaussquad(n, qr); // Obtain quadrature rule

        // Transform nodes and weights to new interval
        Eigen::VectorXd w = qr.weights * M_PI/2;
        Eigen::VectorXd c = ( -M_PI/2 + M_PI/2*(qr.nodes.array() + 1)
            ).matrix();

        // Evaluate g at quadrature nodes c
        Eigen::VectorXd gc = c.unaryExpr(g);

        // Same as I = \sum_{i=1}^n w_i g(c_i)
        double I = w.dot(gc);

        evals.push_back(n); // Save no. of quadrature nodes
        error.push_back(std::abs(I - I_ex)); // Save error
    }

    Eigen::Map<Eigen::VectorXd> numNodes(evals.data(), evals.size());
    Eigen::Map<Eigen::VectorXd> quadErr(error.data(), error.size());

    for (int i=0; i<evals.size(); i++)
        std::cout << numNodes(i) << "\t" << quadErr(i) << std::endl;

}
```

(e) Plot the convergence behaviour for $f(t) = \sinh(t)$ using `smoothIntegrand` and compare with the plot obtained from Sub-problem (b).

SOLUTION:

The plot indicates exponential convergence (being a straight line in lin-log scale).

The convergence is now exponential. The integrand of the original integral belongs to $C^0([-1,1])$ but not to $C^1([-1,1])$, because the derivative of the $\arcsin$ function blows up in ± 1. The change of variable provides a perfectly smooth (analytic) integrand: $x\cos(x)\sinh(\sin x)$. Gauss quadrature enjoys exponential convergence only if the integrand is (at least) $\in C^\infty$ on the closed integration interval. Otherwise, one can only expect algebraic convergence.


Error
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
10^{-13}
0
10
20
30
40
50
No. of quadrature nodes, n
Error