

C++ code 7.13.4: Convenience struct `cvgdata`

```
2 struct cvgdata {
3     long n_{0};           // number of quadrature points
4     double val_{0.0};     // approximate value
5     double err_{-1.0};    // quadrature error
6     double rate_{-1.0};   // estimate for rate of algebraic convergence
7 };
```

Get it on  GitLab ([arccosquad.hpp](#)).

C++ code 7.13.5: Code for `testConvGaussQuad()`

```
2 void testConvGaussQuad() {
3     // Table header
4     std::cout << std::setw(5) << "n" << std::setw(20) << " I_n " << std::setw(30)
5         << " Error " << std::setw(10) << (" rate\n " + std::string(65, '-'))
6         << std::endl;
7
8     // Value obtained by overkill quadrature as reference value
9     constexpr double ref_val = 1.7576137811123187;
10
11    // TODO: (7-13.a) Print a table that allows you to predict the
12    // asymptotic
13    // behaviour of Gauss-Legendre numerical quadrature when approximating
14    // I(f).
15    // START
16    // The integrand as lambda funtion
17    auto integrand = [](double t) {
18        return std::acos(t) * 1.0 / (1.0 + std::exp(t));
19    };
20
21    // For recording errors etc.
22    std::vector<cvgdata> data;
23    // We suspect algebraic convergence. Accordingly use doubling of
24    // number of
25    // quadrature nodes
26    constexpr unsigned int L = 8; // Use at most 2^L quadrature nodes
27    unsigned int n = 2;
28    for (unsigned int l = 1; l <= L; ++l, n *= 2) {
29        const QuadRule qr{gaussquad(n)};
30        // No transformation required: the integral is on [-1,1] anyway.
31        double I = 0.0; // Summation variable for quadrature formula
32        for (unsigned int j = 0; j < n; ++j) {
33            I += qr.weights_[j] * integrand(qr.nodes_[j]);
34        }
35        data.push_back({qr.weights_.size(), I, 0.0, 0.0});
36    }
37    for (unsigned int l = 0; l < data.size(); ++l) {
38        data[l].err_ = std::abs(data[l].val_ - ref_val);
39        if (l > 0) {
40            // Estimate for rate of algebraic convergence
41            data[l].rate_ =
42                (std::log(data[l-1].err_) - std::log(data[l].err_)) / std::log(2.0);
43        }
44    }
45
46    // Output table
47    for (unsigned int l = 0; l < data.size(); ++l) {
48        const cvgdata& item{data[l]};
```