Prof. R. Hiptmair
G. Alberti,
F. Leonardi

AS 2015

ETH Zürich
D-MATH

Numerical Methods for CSE

Problem Sheet 10

Problem 1. Zeros of orthogonal polynomials (core problem)

This problem combines elementary methods for zero finding with 3-term recursions satisfied by orthogonal polynomials.

The zeros of the Legendre polynomial P_n (see [1, Def. 5.3.26]) are the n Gauss points ξ_j^n, $j = 1, \dots, n$. In this problem we compute the Gauss points by zero finding methods applied to P_n. The 3-term recursion [1, Eq. (5.3.32)] for Legendre polynomials will play an essential role. Moreover, recall that, by definition, the Legendre polynomials are $L^2(]-1,1[)$-orthogonal.

(1a) ⚄ Prove the following interleaving property of the zeros of the Legendre polynomials. For all $n \in \mathbb{N}_0$ we have

$$-1 < \xi_j^n < \xi_j^{n-1} < \xi_{j+1}^n < 1, \qquad j = 1, \dots, n-1.$$

HINT: You may follow these steps:

1. Understand that it is enough to show that every pair of zeros (ξ_l^n, ξ_{l+1}^n) of P_n is separated by a zero of P_{n-1}.
2. Argue by contradiction.
3. By considering the auxiliary polynomial $\prod_{j \neq l, l+1}(t - \xi_j^n)$ and the fact that the Gauss quadrature is exact on $\mathcal{P}_{2n-1}$ prove that $P_{n-1}(\xi_l^n) = P_{n-1}(\xi_{l+1}^n) = 0$.
4. Choose $s \in \mathcal{P}_{n-2}$ such that $s(\xi_j^n) = P_{n-1}(\xi_j^n)$ for every $j \neq l, l+1$, and using again that Gauss quadrature is exact on $\mathcal{P}_{2n-1}$ obtain a contradiction.

(1b) ⚀ By differentiating [1, Eq. (5.3.32)] derive a combined 3-term recursion for the sequences $(P_n)_n$ and $(P'_n)_n$.

(1c) Use the recursions obtained in (1b) to write a C++ function

```
void legvals(const Eigen::VectorXd &x, Eigen::MatrixXd
    &Lx, Eigen::MatrixXd &DLx)
```

that fills the matrices `Lx` and `DLx` in $\mathbb{R}^{N\times(n+1)}$ with the values $(P_k(x_j))_{jk}$ and $(P'_k(x_j))_{jk}$, $k = 0,\dots,n$, $j = 0,\dots,N-1$, for an input vector $x \in \mathbb{R}^N$ (passed in `x`).

(1d) We can compute the zeros of P_k, $k = 1,\dots,n$, by means of the secant rule (see [1, § 2.3.22]) using the endpoints $\{-1,1\}$ of the interval and the zeros of the previous Legendre polynomial as initial guesses, see (1a). We opt for a correction based termination criterion (see [1, Section 2.1.2]) based on prescribed relative and absolute tolerance (see [1, Code 2.3.25]).

Write a C++ function

```
MatrixXd gaussPts(int n, double rtol=1e-10, double
    atol=1e-12)
```

that computes the Gauss points $\xi_j^k \in [-1,1]$, $j = 1,\dots,k$, $k = 1,\dots,n$, using the zero finding approach outlined above. The Gauss points should be returned in an upper triangular $n\times n$-matrix.

HINT: For simplicity, you may want to write a C++ function

```
double Pkx(double x, int k)
```

that computes $P_k(x)$ for a scalar x. Reuse parts of the function `legvals`.

(1e) Validate your implementation of the function `gaussPts` with $n = 8$ by computing the values of the Legendre polynomials in the zeros obtained (use the function `legvals`). Explain the failure of the method.

HINT: See Figure 6.

(1f) Fix your function `gaussPts` taking into account the above considerations. You should use the *regula falsi*, that is a variant of the secant method in which, at each step, we choose the old iterate to keep depending on the signs of the function. More precisely, given two approximations $x^{(k)}$, $x^{(k-1)}$ of a zero in which the function f has different signs,

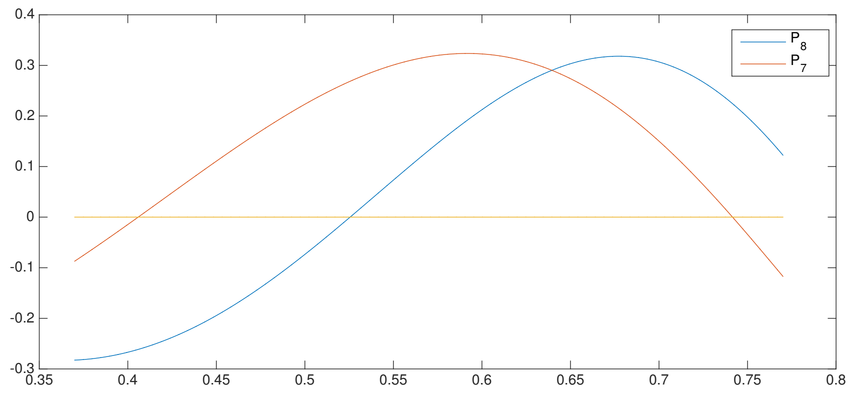

Figure 6: P_7 and P_8 on a part of $[-1,1]$. The secant method fails to find the zeros of P_8 (blue curve) when started with the zeros of P_7 (red curve).

compute another approximation $x^{(k+1)}$ as zero of the secant. Use this as the next iterate, but then chose as $x^{(k)}$ the value $z \in \{x^{(k)}, x^{(k-1)}\}$ for which $\mathrm{sign} f(x^{(k+1)}) \neq \mathrm{sign} f(z)$. This ensures that f has always a different sign in the last two iterates.

HINT: The regula falsi variation of the secant method can be easily implemented with a little modification of [1, Code 2.3.25]:

```
function x = secant_falsi(x0,x1,F,rtol,atol)
fo = F(x0);
for i=1:MAXIT
  fn = F(x1);
  s = fn*(x1-x0)/(fn-fo); % correction
  if (F(x1 - s)*fn < 0)
    x0 = x1;  fo = fn;  end
  x1 = x1 - s;
  if ((abs(s) < max(atol,rtol*min(abs([x0;x1]))]))
    x = x1; return; end
end
```

Problem 2. Gaussian quadrature (core problem)

Given a smooth, odd function $f : [-1, 1] \to \mathbb{R}$, consider the integral

$$I := \int_{-1}^{1} \arcsin(t)\; f(t)\,\mathrm{d}t. \tag{31}$$

We want to approximate this integral using global Gauss quadrature. The nodes (vector `x`) and the weights (vector `w`) of n-point Gaussian quadrature on $[-1, 1]$ can be computed using the provided MATLAB routine `[x,w]=gaussquad(n)` (in the file `gaussquad.m`).

(2a) ⚂ Write a MATLAB routine

```
function   GaussConv(f_hd)
```

that produces an appropriate convergence plot of the quadrature error versus the number $n = 1, \ldots, 50$ of quadrature points. Here, `f_hd` is a handle to the function f.

Save your convergence plot for $f(t) = \sinh(t)$ as `GaussConv.eps`.

HINT: Use the MATLAB command `quad` with tolerance `eps` to compute a reference value of the integral.

HINT: If you cannot implement the quadrature formula, you can resort to the MATLAB function

```
function   I = GaussArcSin(f_hd,n)
```

provided in implemented `GaussArcSin.p` that computes n-points Gauss quadrature for the integral (31). Again `f_hd` is a function handle to f.

(2b) ⚀ Which kind of convergence do you observe?

(2c) ⚁ Transform the integral (31) into an equivalent one with a suitable change of variable so that Gauss quadrature applied to the transformed integral converges much faster.

(2d) ⚂ Now, write a MATLABroutine

```
function  GaussConvCV(f_hd)
```

which plots the quadrature error versus the number $n = 1, \ldots, 50$ of quadrature points for the integral obtained in the previous subtask.

Again, choose $f(t) = \sinh(t)$ and save your convergence plot as `GaussConvCV.eps`.

HINT: In case you could not find the transformation, you may rely on the function

```
function   I = GaussArcSinCV(f_hd,n)
```

implemented in `GaussArcSinCV.p` that applies n-points Gauss quadrature to the transformed problem.

(2e) ⚂ Explain the difference between the results obtained in subtasks (2a) and (2d).

Problem 3. Numerical integration of improper integrals

We want to devise a numerical method for the computation of improper integrals of the form $\int_{-\infty}^{\infty} f(t)dt$ for continuous functions $f : \mathbb{R} \to \mathbb{R}$ that decay sufficiently fast for $|t| \to \infty$ (such that they are integrable on $\mathbb{R}$).

A fist option (T) is the truncation of the domain to a bounded interval $[-b, b], b \leq \infty$, that is, we approximate:

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

and then use a standard quadrature rule (like Gauss-Legendre quadrature) on $[-b, b]$.

(3a) ⚀ For the integrand $g(t) := 1/(1+t^2)$ determine b such that the truncation error E_T satisfies:

$$E_T := \left| \int_{-\infty}^{\infty} g(t)dt - \int_{-b}^{b} g(t)dt \right| \leq 10^{-6} \tag{32}$$

(3b) ⚀ What is the algorithmic difficulty faced in the implementation of the truncation approach for a generic integrand?

A second option (S) is the transformation of the improper integral to a bounded domain by substitution. For instance, we may use the map $t = \cot(s)$.

(3c) ⚁ Into which integral does the substitution $t = \cot(s)$ convert $\int_{-\infty}^{\infty} f(t)dt$?

(3d) ⚀ Write down the transformed integral explicitly for $g(t) := \frac{1}{1+t^2}$. Simplify the integrand.

(3e) Write a C++ function:

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

that uses the transformation from (3d) together with n-point Gauss-Legendre quadrature to evaluate $\int_{-\infty}^{\infty} f(t)dt$. f passes an object that provides an evaluation operator of the form:

```
double operator() (double x) const;
```

HINT: See `quadinf_template.cpp`.

HINT: A lambda function with signature

```
(double) -> double
```

automatically satisfies this requirement.

(3f) Study the convergence of the quadrature method implemented in (3e) for the integrand $h(t) := \exp(-(t-1)^2)$ (shifted Gaussian). What kind of convergence do you observe?

HINT:

$$\int_{-\infty}^{\infty} h(t)dt = \sqrt{\pi} \tag{33}$$

Problem 4. Quadrature plots

We consider three different functions on the interval $I = [0,1]$:

$$\begin{aligned}
&\text{function A:} \quad f_A \in C^\infty(I)\,, \quad f_A \notin \mathcal{P}_k \;\forall\; k \in \mathbb{N}\,;\\
&\text{function B:} \quad f_B \in C^0(I)\,, \quad f_B \notin C^1(I)\,;\\
&\text{function C:} \quad f_C \in \mathcal{P}_{12}\,,
\end{aligned}$$

where $\mathcal{P}_k$ is the space of the polynomials of degree at most k defined on I. The following quadrature rules are applied to these functions:

- quadrature rule A, global Gauss quadrature;

- quadrature rule B,   composite trapezoidal rule;
- quadrature rule C,   composite 2-point Gauss quadrature.

The corresponding absolute values of the quadrature errors are plotted against the number of function evaluations in Figure 7. Notice that only the quadrature errors obtained with an even number of function evaluations are shown.

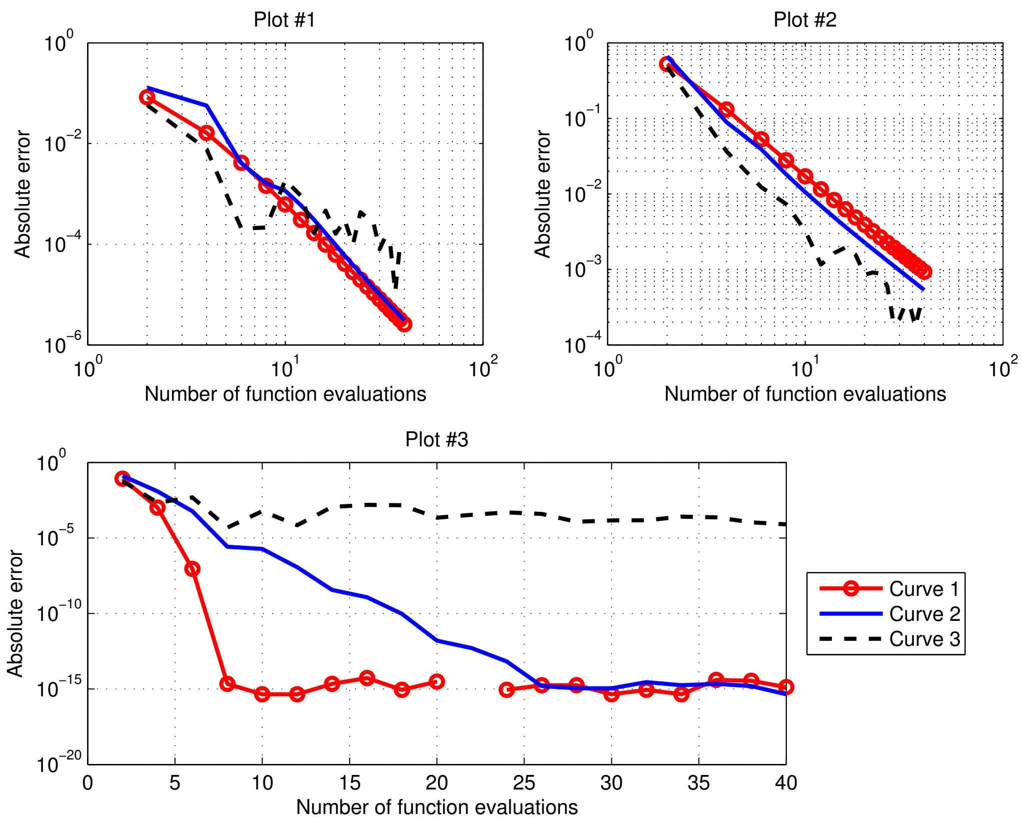

Figure 7: Quadrature convergence plots for different functions and different rules.

(4a) ⚂ Match the three plots (plot #1, #2 and #3) with the three quadrature rules (quadrature rule A, B, and C). Justify your answer.

HINT:: notice the different axis scales in the plots.

(4b) ⚂ The quadrature error curves for a particular function f_A, f_B and f_C are plotted in the same style (curve 1 as red line with small circles, curve 2 means the blue solid line, curve 3 is the black dashed line). Which curve corresponds to which function (f_A, f_B, f_C)? Justify your answer.

Issue date: 19.11.2015
Hand-in: 26.11.2015 (in the boxes in front of HG G 53/54).

Version compiled on: November 19, 2015 (v. 1.0).

References

[1] R. Hiptmair. *Lecture slides for course "Numerical Methods for CSE"*. http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf. 2015.