MATH/CS 5466 · NUMERICAL ANALYSIS

Problem Set 5

Posted Friday 8 April 2016. Due Monday 18 April 2016 (5pm).
Please complete all 5 problems (total of 125 points).

1. [25 points]

   When tasked with computing a definite integral $\int_a^b f(x)\,\mathrm{d}x$, one is not forced to apply some quadrature rule directly to f. Sometimes a bit of forethought can lead to a more effective approach.

   (a) Does the standard error bound derived in class for the composite trapezoid rule give any insight about the performance of that method applied to the Fresnel integral

   $$\int_0^1 \frac{\sin(x)}{\sqrt{x}}\,\mathrm{d}x\,?$$

   (b) One can compute this Fresnel integral by expanding $\sin(x)$ in a Taylor series about $x = 0$ to obtain

   $$\int_0^{\varepsilon} \Big(\frac{x - \frac{1}{3!}x^3 + \cdots}{\sqrt{x}}\Big)\,\mathrm{d}x + \int_{\varepsilon}^1 \frac{\sin(x)}{\sqrt{x}}\,\mathrm{d}x.$$

   To approximate this quantity, truncate the Taylor series after m terms and compute the resulting approximation to the first integral exactly; approximate the second integral using the composite trapezoid rule. Estimate the accuracy of this procedure as a function of m, ε, and the number of subintervals used in the composite trapezoid rule for the second integral.

   (c) Construct a function $g(x)$ such that the integral in part (a) can be effectively computed via

   $$\int_0^1 \Big(\frac{\sin(x)}{\sqrt{x}} - g(x)\Big)\,\mathrm{d}x + \int_0^1 g(x)\,\mathrm{d}x,$$

   where the first integral can be computed accurately with the composite trapezoid rule and the second integral can be computed exactly without need for numerical quadrature.

   [Bulirsch and Stoer]

2. [25 points]

   The gamma function is defined as

   $$\Gamma(z) = \int_0^\infty x^{z-1}\mathrm{e}^{-x}\,\mathrm{d}x.$$

   (a) Use the composite Simpson's rule to evaluate $\Gamma(5)$, $\Gamma(10)$, and $\Gamma(.5)$ to five digits of accuracy. You may use MATLAB's built-in `gamma` function to verify your answer. Please report the number of integrand evaluations required. In particular, design a way to reliably compute $\Gamma(.5)$ using fewer than 1000 function evaluations. (Consider further partitioning the domain into subdomains.)

   The orthogonal polynomials on $[0,\infty)$ with weight function $w(x) = \mathrm{e}^{-x}$ are called Gauss–Laguerre polynomials. These polynomials can be constructed from the recurrence

   $$L_0(x) = 1, \qquad L_1(x) = x - \alpha_0 = x - 1, \qquad L_{k+1}(x) = (x-\alpha_k)L_k(x) - \beta_k L_{k-1}(x), \quad k = 1,2,\ldots,$$

   where $\alpha_k = 2k+1$ for $k = 0,1,\ldots$, $\beta_0 = 1$, and $\beta_k = k^2$ for $k = 1,2,\ldots$.

   (b) Use your answer from Question 3 of Problem Set 4 to design a MATLAB function

   ```
   [x,w] = gauss_lag(n)
   ```

that computes the nodes `x` and weights `w` for $n+1$-point Gauss–Laguerre quadrature from the corresponding Jacobi matrix $\mathbf{J}$. Recall that the weights are given by

$$w_j = \beta_0(\mathbf{v}_j)_1^2,$$

where $(\mathbf{v}_j)_1^2$ is the square of the first entry of the eigenvector $\mathbf{v}_j$ of $\mathbf{J}$ corresponding to eigenvalue x_j. (Here $\mathbf{v}_j$ should be normalized so that $\|\mathbf{v}_j\|_2 = 1$, which is automatically imposed upon the eigenvectors returned by MATLAB's `eig` command.)

(c) Produce a *single plot* comparing the nodes of all $n+1$-point Gauss–Laguerre rules for $n = 0, \ldots, 25$. For example, you can plot each set of nodes with

```
plot(x, n*ones(n+1,1), '.','markersize',24)
```

which displays the nodes horizontally at vertical level n. (Notice how the largest node, x_n, grows as n increases, as the rule integrates over the unbounded interval $[0, \infty)$.)

(d) Compute $\Gamma(5)$, $\Gamma(10)$, and $\Gamma(.5)$ using this $n+1$-point Gauss–Laguerre rule for $n = 5$. (Each of these integrals will only require six $f(x)$ evaluations.)

(d) How does the accuracy of Gauss–Laguerre quadrature compare to that obtained by the composite Simpson's rule? Give pros and cons of each method for this problem.

3. [25 points]

How might the implementer of a mathematical function library (or even hardware designer) implement a function like `sqrt` using only basic operations (addition, subtraction, multiplication, division)? Various clever approaches exist. This question (from problems in Gautschi's book) considers several options using Newton's method.

(a) Given some fixed $A > 0$, one can compute $x_* = \sqrt{A}$ via the root-finding problem

$$x^2 - A = 0.$$

Explain why Newton's method applied to the first equation converges for any initial value $x_0 > 0$. Be as rigorous as possible. (Hint: drawing a few sketches will help you understand what is going on much more quickly than wrangling algebraic expressions.)

(b) Alternatively, one might compute $x_* = \sqrt{A}$ via the root-finding problem

$$\frac{A}{x^2} - 1 = 0.$$

Show that the initial set of positive initial guesses $x_0 > 0$ can be partitioned into three sets:

$$\begin{aligned} x \in (0, \alpha) \quad &\Longrightarrow \quad x_k \to \sqrt{A} \text{ and all } x_k > 0; \\ x \in (\alpha, \beta) \quad &\Longrightarrow \quad x_k \to \pm\sqrt{A},\ x_k \text{ do not all have the same sign;} \\ x \in (\beta, \infty) \quad &\Longrightarrow \quad x_k \text{ diverges.} \end{aligned}$$

(You should specify formulas for α and β as a function of A.)

(c) Repeat parts (a) and (b) for the analogous approaches to the cube root,

$$x^3 - A = 0, \qquad \frac{A}{x^3} - 1 = 0.$$

In generalizing part (b), simply identify the interval $(0, \alpha)$ on which the iterates are all positive and converge to $\sqrt{A}$.

(d) One can also compute to compute $x_* = \sqrt{A}$ via the fixed point iteration

$$x_{k+1} = \frac{x_k(x_k^2 + 3a)}{3x_k^2 + a}.$$

(i) Show that *if the sequence converges to* $\sqrt{A}$, it convergence *cubically*.
(ii) Determine the *asymptotic error constant*.
(iii) Discuss (using graphical or numerical support) the convergence of this method for $x_0 > 0$.

[Gautschi]

4. [25 points]

Recall the derivation of Newton's method: A function $f \in C^2(\mathbb{R})$ was expanded in a Taylor series up to first order. Here we investigate the algorithm obtained by taking one more term in the Taylor series. Assume $f \in C^3(\mathbb{R})$.

(a) Derive an algorithm analogous to Newton's method but based on the first three terms of the Taylor series for $f(x_*)$ expanded at $f(x_k)$, rather than just the first two. This method should include evaluations of both $f'(x_k)$ and $f''(x_k)$.

(b) At each iteration, Newton's method approximates the root of f by the root of the line tangent to f at x_k. Your new algorithm should approximate f by a parabola. Draw this parabola, together with the tangent line used by Newton's method, for the function $f(x) = \sin(x)e^x$ at the point $x_0 = 1.25$. (Please produce a careful MATLAB plot, not a hand-drawn sketch.)

(c) The parabola in question will often have two roots. Describe which of these roots should be selected at each iteration.

(d) Under what circumstances will this parabola have no roots?

(e) Implement this algorithm in MATLAB, in a form similar to the code `newton.m` discussed in class.

(f) Compare the results of your method with those for `newton.m` on $f(x) = \sin(x)e^x$ with $x_0 = 1.25$.

(g) Conjecture about the convergence rate of this method when $f''(x_*) \neq 0$ and x_0 is sufficiently close to x_*. Why isn't this algorithm more famous than Newton's method?

(h) How do you expect this method to perform near a double root? Why? Compare your iteration to Newton's method for $f(x) = x^2e^x$ with $x_0 = 1$.

(*Halley's method* is a more famous alternative to Newton's method, that is similar to the algorithm described in this problem, but more robust. See Problem 43 on page 301 of Gautchsi's book.)

5. [25 points]

Polynomial root-finding is an important special case of the general root-finding problem. Thus it is no surprise that innumerable specialized algorithms have been proposed for computing the roots of a polynomial. The method outlined in this problem is particularly appealing, for it is guaranteed to find n roots of a degree-n polynomial. Indeed, it is the basis for MATLAB's `roots` command.

Suppose we seek the zeros of the degree-n polynomial

$$p(x) = c_0 + c_1x + \cdots + c_nx^n.$$

(a) Show that $p(x) = 0$ if and only if x is an eigenvalue of the *companion* matrix

$$\mathbf{C}_n = \begin{bmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ -\dfrac{c_0}{c_n} & -\dfrac{c_1}{c_n} & \cdots & -\dfrac{c_{n-2}}{c_n} & -\dfrac{c_{n-1}}{c_n} \end{bmatrix}$$

with eigenvector $\mathbf{v}(x) = [1, x, \ldots, x^{n-1}]^T$.

(b) Verify your results numerically with the polynomial

$$p(x) = (x-1)(x-2)(x-3).$$

(c) Repeat this experiment for the degree-24 *Wilkinson's polynomial*:

$$p(x) = (x-1)(x-2)\cdots(x-24).$$

(You can compute the coefficients $c_0, \ldots, c_{24}$ in MATLAB via the command `poly(1:24)`.) Do you correctly compute the roots $x = 1, 2, \ldots, 24$?

(d) Suppose that we have a polynomial represented in the basis of Chebyshev polynomials,

$$p(x) = a_0 T_0(x) + a_1 T_1(x) + \cdots + a_n T_n(x),$$

where, as usual,

$$T_0(x) = 1, \quad T_1(x) = x, \qquad T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad k = 1, 2, \ldots.$$

Generalize the notion of a companion matrix to this setting, showing how the roots of p can be found as eigenvalue of some matrix $\mathbf{G}_n$ that encodes the coefficients $a_0, \ldots, a_n$. What are the corresponding eigenvectors?

(We have used the notation "$\mathbf{G}_n$" in honor of Jack Good, a Bletchley Park codebreaker with Alan Turing during World War Two, and (later) a Virginia Tech Statistics professor, who was one of the first to work out this matrix.)

(e) Verify your algorithm for the polynomial

$$p(x) = -\frac{1}{8}T_0(x) + \frac{1}{2}T_1(x) - \frac{1}{2}T_3(x) + \frac{1}{8}T_4(x),$$

which has roots $-1, 1, 2, 3$.

(f) How is this result connected to your answer from Question 3 of Problem Set 4 ?