Worksheet 9: Integration Methods

June 26, 2017

General Remarks

- The deadline for handing in the worksheets is **Monday, July 3rd, 2017, 12:00 noon**.
- For this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to your tutor:
  - Johannes Zeman `zeman@icp.uni-stuttgart.de` (Tue 15:45–17:15)
  - Michael Kuron `mkuron@icp.uni-stuttgart.de` (Wed 15:45–17:15)
  - Johannes Zeman `zeman@icp.uni-stuttgart.de` (Thu 14:00–15:30)
- Please try to only hand in a single file that contains your program code for all tasks. If you are asked to answer questions, you should do so in a comment in your code file. If you are asked for graphs or figures, it is sufficient if your code generates them. You may as well hand in a separate PDF document with all your answers, graphs and equations.
- The worksheets are to be solved in groups of two or three people.

In the lecture, you learned about different numerical integration methods, of which you already implemented one in the previous worksheet. In this worksheet, your job will be to implement and compare the remaining methods.

Task 9.1: Integration Schemes with Equidistant Support Points (3 points)

- **9.1.1** (1 point) Implement a Python function `midpoint(f,a,b,n)` that approximates the integral $\int_a^b f(x)\,dx$ using the composite midpoint rule with n support points on the interval (a, b). Do not use functions from `scipy.integrate` or similar modules to do so.
- **9.1.2** (1 point) Implement a Python function `simpson(f,a,b,n)` that approximates the integral $\int_a^b f(x)\,dx$ using the composite Simpson rule with n support points on the interval $[a, b]$. Do not use functions from `scipy.integrate` or similar modules to do so.

Hints Note that the parameter n is the number of support points, and thus – depending on the integration method – *not necessarily* the number of subintervals the interval $[a, b]$ is split into:

- When using the composite midpoint rule, the support points x_i are *centered* on subintervals of width h, implying that the number of support points indeed equals the number of subintervals. It therefore follows that $(b-a) = nh$, and $x_i = a+h(i+\frac{1}{2})$ with $i = 0, 1, \ldots, n-1$.
- When using the composite trapezoid or Simpson rule, the support points x_i lie *on the edges* of the subintervals, and hence, the number of subintervals is one less than the number of support points. It follows that $(b-a) = (n-1)h$ and $x_i = a+ih$ with $i = 0, 1, \ldots, n-1$.
- The composite Simpson rule only works for an odd number of support points.

- **9.1.3** (1 point) Implement a Python function `romberg(f,a,b,n)` that approximates the integral $\int_a^b f(x)\,dx$ using the Romberg scheme with n support points on the interval $[a, b]$. Here, you can make use of the example script provided on the lecture website (section "Vorlesung"). However, you should not just copy and paste the function as is, but carefully read and understand what is done in every step. Use NumPy arrays instead of lists in the computations *without* making use of NumPy's `append()` method. The example script imports the required modules with wildcards (`*`), you should *not* do that in your own solution. Also note that the maximum number of support points used in the example script is actually `int(log2(n))+1`.

Task 9.2: Convergence of the Different Integration Schemes (3 points)

- **9.2.1** (1 point) Implement a Python function `erf(integrator,x,n)` that computes the value of the error function

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2}\,d\tau \tag{1}$$

at the point x using n support points. The parameter `integrator` is used to pass a function which performs the required integration. For example, if the error function is to be computed at $x = 1$ using the composite midpoint rule with 3 support points, this should be possible by calling `erf(integrator=midpoint, x=1.0, n=3)`.

Hint If you want, you can use an anonymous `lambda` function to define the integrand.

- **9.2.2** (1 point) Verify your implementations from task 9.1 by evaluating the error function $\operatorname{erf}(x)$ at 100 equidistant points on the interval $[0, 2]$ using the different integration schemes with $n = 3$ support points. Plot the resulting curves together with a reference curve obtained from `scipy.special.erf()`.
- **9.2.3** (1 point) Compute a reference value $e_{\text{ref}} = \operatorname{erf}(1)$ using `scipy.special.erf(1)`. Create a loglog plot of the absolute deviations of `erf(integrator, 1.0, n)` from e_{ref} when computing $\operatorname{erf}(1)$ for $n = 2^m + 1$ with $m = 1, 2, \ldots, 6$ using the different integration schemes. Depending on the integration scheme, how do the deviations scale with n?

Task 9.3: Gauss-Legendre Quadrature (2 points)

- **9.3.1** (1 point) Implement a Python function `gaussian_quadrature(f,a,b,n)` that approximates the integral $\int_a^b f(x)\,dx$ using the Gaussian quadrature method with n support points on the interval (a, b). Do not use functions from `scipy.integrate` or similar modules to do so.

Hints Use the function `numpy.polynomial.legendre.leggauss()` to compute the location of the support points and the corresponding coefficients.

- **9.3.2** (1 point) Similar to task 9.2.3, compute $\operatorname{erf}(1)$ for $n = 1, 2, \ldots, 10$ support points using the Gaussian quadrature method and plot the absolute deviation from e_{ref} with respect to n in a graph with a logarithmically scaled y-axis (semilogy). How does the deviation scale with n?

Task 9.4: Estimating π (2 points)

The value of the number π can be obtained from the following equation:

$$\frac{\pi}{4} = \int_0^1 \sqrt{1 - x^2}\, dx \tag{2}$$

- **9.4.1** (1 point) Implement a Python function `pi(integrator,n)` that numerically approximates the value of π according to Eq. (2). As in task 9.2.1, the parameter `integrator` shall be used to pass a function which performs the required integration using n support points.
- **9.4.2** (1 point) Compute estimates of π according to Eq. (2) using $n = 2^m+1$ with $m = 1, 2, \ldots, 6$ support points to approximate the integral with each of the previously implemented integration schemes (midpoint, Simpson, and Romberg integration, as well as Gaussian quadrature). Use `np.pi` as a reference value und plot the deviations of the obtained estimates as a function of n in a loglog plot. Compare the behaviors of the different integrators to the ones you plotted in tasks 9.2.3 and 9.3.2. Describe the differences and explain their origin!