PURDUE UNIVERSITY · CS 31400
NUMERICAL ANALYSIS

HOMEWORK
*David F. Gleich*
*March 23, 2021*

Please answer the following questions in complete sentences in a typed manuscript and submit the solution on Gradescope by April 5th around 5am like our usual deadline.

## Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

## Problem 1: Gautschi Exercise 3.29 + more

1. Derive the 2-point Gauss-Hermite quadrature rule.
2. Example application (from http://ice.uchicago.edu/2012_presentations/Faculty/Judd/Quadrature_ICE11.pdf). An investor holds one bond that will be worth 1 in the future and equity whose value is $Z$ where $\log Z \sim N(\mu, \sigma^2)$. (So this means that the log of the value of the expected utility is a normally distributed random variable.) The expected utility is the random number $U = f(1 + Z)$. where $f$ is a utility function, we'll use $f(x) = x^{1+\gamma}/(1 + \gamma)$, where $\gamma < 0$. (This is a concave utility function because having more money doesn't give you all that much more utility.) We'll use $\gamma = -0.5$. Suppose also that $\mu = 0.15$ and $\sigma = 0.25$. We want to find the expected utility to the investor! This involves evaluating the integral
$$E[U] = \int_{-\infty}^{\infty} f(1 + e^x) e^{-(x-\mu)^2/(2\sigma^2)}\, dx.$$
Write a compute program to use Gauss-Hermite quadrature to approximate the value of this integral. You need to justify the number of points you use for the approximation.

## Problem 2: Multivariate quadrature

In this problem, we will investigate multivariate quadrature for integrals such as
$$I = \int_{-1}^{1} \int_{-1}^{1} f(x, y)\, dxdy$$
using *tensor product* rules. Let $t_i$ and $w_i$ be the nodes and weights of an $n$-point 1-dimensional Gauss-Legendre rule. Then the multidimensional quadrature rule is:
$$I \approx \sum_{i=1}^{n} \sum_{j=1}^{n} f(t_i, t_j) w_i w_j.$$
We can derive this as follows:
$$I \approx \int_{-1}^{1} \sum_{i=1}^{n} f(t_i, y) w_i dy \approx \sum_{i=1}^{n} \sum_{j=1}^{n} f(t_i, t_j) w_i w_j.$$

Of course, this makes it clear we don't have to use the same number of points to integrate in $x$ and $y$! So in general, let $t_i^{(x)}, w_i^{(x)}$ be an $N_x$-point Gauss-Legendre quadrature rule for the $x$-variable and $t_i^{(y)}, w_i^{(y)}$ be an $N_y$-point Gaussian quadrature for the $y$ variable.

1. Implement a computer code to perform two-dimensional Gauss-Legendre quadrature using this type of construction. Your code should allow a user to input $N_x$ and $N_y$ to determine the number of points in each variable.
2. Use your code to estimate the integrals using 10 points in each dimension.
   - $f(x,y) = x^2 + y^2$
   - $f(x,y) = x^2 y^2$
   - $f(x,y) = exp(x^2 + y^2)$
   - $f(x,y) = (1 - x^2) + 100 * (y - x^2)^2$
3. *This is an open ended question that requires you to investigate.* You can also find the answer in many textbooks, but if you do so, make sure you document your sources and demonstrate the effect that is claimed. We saw in class that an $n$-point Gaussian quadrature rule exactly integrated polynomials of up to degree $2n - 1$. In this problem, I want you to generate a conjecture about the degree of exactness of multidimensional Gaussian quadrature. You should use your code from part 1, along with carefully constructed examples, to support a statement such as:

   ```
   My evidence suggests that 2d Gauss-Legendre quadrature will exactly
   integrate two-dimensional functions $f(x,y)$ when ...
   ```

   Here are some helpful ideas that may play a role.

   - The total degree of a multidimensional polynomial is the maximum of the sum of degrees of each term. So $f(x,y) = x^2 y^2$ has total degree 4.
   - Another type of degree is the largest degree in each variable, so $f(x,y) = x^2 y^2$ involves polynomials of degree 2 only.

## Problem 3: Monte Carlo, Gaussian-Legendre, and Clenshaw-Curtis Quadrature

In this problem, we'll study how different quadrature methods converge on a variety of problems. For a technical paper on this idea, see Trefethen, Is Gaussian Quadrature better than Clenshaw-Curtis? SIAM Review, 2008. In this problem, we'll be studying and reproducing Fig. 2 from that paper.

The functions are all defined on the region $[-1, 1]$ and are:

- $f_1(x) = x^{20}$
- $f_2(x) = e^{-x^2}$
- $f_3(x) = e^{-1/(x^2)}$
- $f_4(x) = e^x$
- $f_5(x) = 1/(1 + 16x^2)$
- $f_6(x) = |x|^3$

The quadrature methods are:

- **Monte Carlo quadrature.** Monte Carlo quadrature is a randomized method. We simply guess $n$ points between $[-1, 1]$ uniformly at random and then take the average of all the function values. For instance, the following Julia code evaluates a Monte Carlo approximation

  ```
  function montecarlo(f::Function, n::Int, a::Float64, b::Float64)
      assert(a < b)
  ```

```
    xi = (b-a)*rand(n)+a
    fi = mean(f(xi))
end
```

- **Clenshaw-Curtis quadrature.** This quadrature uses Chebyshev points of the second kind to build an interpolatory quadrature formula instead of uniformly spaced points (as is common in Newton-Cotes quadrature). It just so happens that there is an incredibly elegant method to compute the weights associated with this quadrature based on the Fast-Fourier transform. See Trefethen's paper above for a 6-line Matlab code that implements Clenshaw-Curtis quadrature.
- **Gauss-Legendre quadrature.** In Gauss-Legendre quadrature, we pick the quadrature nodes and weights together. This gives even more accuracy. To find these nodes and weights, we must evaluate the eigenvalues and one component of each eigenvector of the Jacobi matrix associated with the Legendre orthogonal polynomials. The Jacobi matrix for these polynomials is easy:

$$\boldsymbol{J} = \begin{bmatrix} 0 & \sqrt{1/(4-1/(1^2))} & 0 & 0 & 0 & \dots \\ \sqrt{1/(4-1/(1^2))} & 0 & \sqrt{1/(4-1/(2^2))} & 0 & 0 & \dots \\ 0 & \sqrt{1/(4-1/(2^2))} & 0 & \sqrt{1/(4-1/(3^2))} & 0 & \dots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{bmatrix}$$

  The size of the matrix should be $n \times n$ if you want an $n$-point formula. To get the nodes, we just look at the eigenvalues of the matrix. To get the weights, we need to get the first component of each eigenvector, and square it. (Hint: see Trefethen's paper for a simple Matlab code.)

1. By whatever method you want, determine the exact values of these 6 integrals. (Hint: Wolfram Alpha is okay!)
2. Write a program to create the Jacobi matrix for Gauss-Legendre quadrature and show the eigenvalues of the matrix for $n = 11$.
3. Implement a computer program for Clenshaw-Curtis quadrature. (Hint: you can copy Trefethen's routines, with attribution) but you must explain the steps. Julia implementations are advisable too.
4. Implement a computer program for Gauss-Legendre quadrature.
5. Note that the Monte Carlo method is a randomized algorithm, so the result will change if you do it again. Each run is called a trial or sample. Use a computer implementation of Monte Carlo integration to how much the values in a Monte Carlo integration can vary between one trial and the next. Compute the variance for $n = 100$ and 1000 samples for each of the above functions.
6. Prepare 6 figures like Trefethen had in his paper for the 6 functions. Except use Monte Carlo integration instead of Newton-Cotes.
7. Empirically determine how computing the Gaussian Quadrature nodes and weights scales as a function of $n$, the number of points, in terms of CPU time. (Hint, consider $n$ between 100 and 1000.) You should be able to justify your final scaling constant in terms of the runtime of a known algorithm.
8. (Optional) There are some recent developments in *fast quadrature* methods that produce the nodes and weights much more quickly. In Julia, these are implemented in the `FastGaussQuadrature` package and the `gausslegendre` function and in Matlab, the function

`legpts.m` from Chebfun implements them (an old simple version is here: http://www.mathworks.com/matlabcentral/fileexchange/23972-chebfun-v4-old-version--please-download-current-version-instead/content/chebfun/legpts.m Determine the simplest equation that describes the empirical scaling for these routines to find the quadrature points and weights. (Hint, consider $n$ between 1000 and 100000.)

## Problem 4: Gautschi Exercise 4.1

The following sequences converge to 0 as $n \to \infty$:

- $v_n = n^{-10}$
- $w_n = 10^{-n}$
- $x_n = 10^{-n^2}$
- $y_n = n^{10}3^{-n}$
- $z_n = 10^{-3 \cdot 2^n}$

Indicate the type of convergence for each sequence in terms of

- Sublinear
- Linear
- Superlinear
- Quadratic
- Cubic
- None of the above

## Problem 5: Implementing a core routine

1. Using any method we've seen to solve a scalar nonlinear equation (bisection, false position, secant), develop a routine to compute $\sqrt{x}$ using only addition, subtraction, multiplication, and division (and basic control structures) to numerical precision. (Use double-precision.)
2. Compare the results of your method to the Matlab/Julia/Python function `sqrt`. Comment on any differences that surprise you.