Gaussian Quadrature

1. Constructing Quadrature Rules
   - degree of precision
   - the method of undetermined coefficients
2. Gaussian Quadrature
   - conditions on polynomials
   - orthogonal polynomials
   - Gauss-Legendre quadrature
3. Making Gauss Quadrature Rules
   - reduction to an eigenvalue problem

MCS 471 Lecture 27
Numerical Analysis
Jan Verschelde, 25 October 2021

Gaussian Quadrature

1. Constructing Quadrature Rules
   - degree of precision
   - the method of undetermined coefficients

2. Gaussian Quadrature
   - conditions on polynomials
   - orthogonal polynomials
   - Gauss-Legendre quadrature

3. Making Gauss Quadrature Rules
   - reduction to an eigenvalue problem

numerical integration with quadrature rules

Given a function $f(x)$ over an interval $[a, b]$,
our problem is to approximate the definite integral over f over $[a, b]$,
by a weighted sum of function values:

$$\int_a^b f(x)dx \approx w_1 f(x_1) + w_2 f(x_2) + \cdots + w_n f(x_n).$$

The *quadrature rule* is defined by

- interpolation points $x_i \in [a, b]$, $x_1 < x_2 < \cdots < x_n$; and
- weights w_i to multiply the function values with.

degree of precision

The cost of a quadrature rule is determined by the number of function values, or equivalently, the number of interpolation points.

Definition

A quadrature rule has *degree of precision* d
if the rule integrates all polynomial of degree d or less exactly.

Because $\int_a^b$ is a linear operator:

$$\int_a^b c_d x^d + \cdots + c_1 x + c_0 dx = \int_a^b c_d x^d dx + \cdots + \int_a^b c_1 x dx + \int_a^b c_0 dx,$$

it suffices to compute the degree of precision for the basis functions.

Gaussian Quadrature

1. Constructing Quadrature Rules
   - degree of precision
   - the method of undetermined coefficients

2. Gaussian Quadrature
   - conditions on polynomials
   - orthogonal polynomials
   - Gauss-Legendre quadrature

3. Making Gauss Quadrature Rules
   - reduction to an eigenvalue problem

the method of undetermined coefficients

Problem:
Construct a 3-point integration formula over $[-h, +h]$, for $h > 0$, evaluate at $-h$, 0, and $+h$. Determine the weights so the degree of precision is as high as possible.

Answer: Setup the conditions imposed by the degree of precision.
Let a, b, and c be the weights in $a f(-h) + b f(0) + c f(+h)$.

$$
\begin{array}{lrcccl}
f = 1 : & \int_{-h}^{+h} 1 dx & = & 2h & = & a + b + c \\
f = x : & \int_{-h}^{+h} x dx & = & 0 & = & a(-h) + b0 + c(+h) \\
f = x^2 : & \int_{-h}^{+h} x^2 dx & = & \frac{2h^3}{3} & = & a(-h)^2 + b0^2 + c(+h)^2
\end{array}
$$

Then we solve for a, b, and c.

computing the weights

We have to solve

$$\left\{ \begin{array}{rcl} a + b + c & = & 2h \\ -a + c & = & 0 \\ a + c & = & 2h/3 \end{array} \right.$$

The solution is $a = h/3 = c$, $b = 4h/3$.

$$\int_{-h}^{h} f(x)dx \approx h\left(\frac{1}{3}f(-h) + \frac{4}{3}f(0) + \frac{1}{3}f(+h)\right).$$

This rule is a specific instance of Simpson's rule.

In L-25, we used `SymPy` to derive this rule for $[a, b]$, with function values at a, $(a+b)/2$, and b.

the midpoint rule (again ...)

In the previous example, the interpolation points were given.

We can obtain a higher degree of precision if in the conditions the interpolation are variable as well.

Solving the exercise below will give the midpoint rule.

Exercise 1:

Consider $\displaystyle\int_a^b f(x)dx \approx w_1 f(x_1)$.

Determine w_1 and x_1 so the degree of precision is as high as possible.

a second exercise

Exercise 2:

Consider the quadrature rule

$$\int_{-2a}^{2a} f(x)dx \approx w_1 f(-a) + w_2 f(a), \quad \text{for} \quad a > 0.$$

Determine the weights w_1 and w_2 so that the rule has the highest possible algebraic degree of precision.

Gaussian Quadrature

conditions on polynomials

We seek to determine the interpolation points so polynomials of degree higher than n will be integrated exactly.

Denote $q(x) = (x - x_0)(x - x_1) \cdots (x - x_{n-1})$.

We can write every polynomial f of degree higher than n as

$$f(x) = p_n(x) + q(x)r(x), \quad \deg(p_n) = n, \; p_n(x_i) = f(x_i),$$

and $q(x)r(x)$ contain the higher order terms:

$$r(x) = r_0 + r_1 x + r_2 x^2 + \cdots + r_k x^k,$$

so that $\deg(f) = n + k$.

The quadrature rule will be $\displaystyle\int_a^b p(x) dx$.

conditions on polynomials

The condition to integrate f exactly is

$$\int_a^b f(x)dx = \int_a^b p(x)dx + \underbrace{\int_a^b q(x)r(x)dx}_{=0}.$$

As $r(x) = r_0 + r_1 x + \cdots + r_k x^k$ and $\displaystyle\int_a^b$ is a linear operator, the conditions are equivalent to:

$$\int_a^b q(x)x^i dx = 0, \quad i = 0, 1, \ldots, k,$$

which is a necessary and sufficient condition.

Gaussian Quadrature

1. Constructing Quadrature Rules
   - degree of precision
   - the method of undetermined coefficients

2. Gaussian Quadrature
   - conditions on polynomials
   - orthogonal polynomials
   - Gauss-Legendre quadrature

3. Making Gauss Quadrature Rules
   - reduction to an eigenvalue problem

orthogonal polynomials

$$\int_a^b q(x)x^i dx = 0, \quad i = 0, 1, \ldots, k,$$

means that $q(x)$ is orthogonal to all x^i,
with respect to the inner product

$$\langle f, g \rangle = \int_a^b f(x)g(x)dx.$$

As $\deg(q) = n$, the highest k can go is $n - 1$.

With orthogonal polynomials we can reach a precision of degree $2n - 1$.

Legendre and Chebyshev polynomials

Legendre polynomials: $[a, b] = [-1, +1]$ follow a recursion:

$L_0(x) = 1, \;\; L_1(x) = x, \;\; (n+1)L_{n+1}(x) - (2n+1)xL_n(x) + nL_{n-1}(x) = 0.$

Gauss-Chebyshev quadrature has inner product:

$$\langle f, g \rangle = \int_{-1}^{+1} \frac{f(x)g(x)}{\sqrt{1-x^2}} dx,$$

where the weight function is $1/\sqrt{1-x^2}$.

construction of Gaussian quadrature rules

Three steps to make a Gaussian quadrature rule with n points:

1. Construct the orthogonal polynomial $q(x)$ of degree n.
2. The roots of q are the interpolation points of the rule.
3. The weights are integrals of the Lagrange polynomials.

Gaussian Quadrature

1. Constructing Quadrature Rules
   - degree of precision
   - the method of undetermined coefficients

2. Gaussian Quadrature
   - conditions on polynomials
   - orthogonal polynomials
   - Gauss-Legendre quadrature

3. Making Gauss Quadrature Rules
   - reduction to an eigenvalue problem

Legendre polynomials

The Legendre polynomials are defined by

$$L_0(x) = 1, \ \ L_1(x) = x, \ \ (n+1)L_{n+1}(x) - (2n+1)xL_n(x) + nL_{n-1}(x) = 0.$$

We turn this into an iterative algorithm:

$$L_{n+1}(x) = \frac{1}{n+1} \left(\ (2n+1)\, x\, L_n(x) - n\, L_{n-1}(x) \ \right).$$

To compute the Legendre polynomial of degree $d > 1$:

1. $L_0 = 1$; $L_1 = x$
2. for n from 2 to d do

$$L_n(x) = \frac{1}{n} \left(\ (2n-1)\, x\, L_{n-1}(x) - (n-1)\, L_{n-2}(x) \ \right).$$

defining Legendre polynomials with SymPy

```
using SymPy
x = Sym("x")

"""
    legendre(d::Int)

returns the Legendre polynomial of degree d,
as a SymPy expression.
"""
```

the function `legendre`

```
function legendre(d::Int)
    if d == 0
        return 1
    elseif d == 1
        return x
    end
    L0 = 1
    L1 = x
    L2 = 0
    for n = 2:d
        L2 = expand(((2*n-1)*x*L1 - (n-1)*L0)/n)
        (L0, L1) = (L1, L2)
    end
    return L2
end
```

the first six Legendre polynomials

```
L(0) = 1
L(1) = x
L(2) = 3*x^2/2 - 1/2
L(3) = 5*x^3/2 - 3*x/2
L(4) = 35*x^4/8 - 15*x^2/4 + 3/8
L(5) = 63*x^5/8 - 35*x^3/4 + 15*x/8
```

To extract the coefficients, we use array comprehensions:

```
for d=1:5
    Ld = legendre(d)
    cff = [Ld.coeff(x, k) for k=0:d]
    nbr = [Float64(c) for c in cff]
end
```

The numerical coefficients are input for a numerical root finder.

the first six Legendre coefficient vectors

```
Symbolic coefficients :
L(1) : Sym[0, 1]
L(2) : Sym[-1/2, 0, 3/2]
L(3) : Sym[0, -3/2, 0, 5/2]
L(4) : Sym[3/8, 0, -15/4, 0, 35/8]
L(5) : Sym[0, 15/8, 0, -35/4, 0, 63/8]
Numeric coefficients :
L(1) : [0.0, 1.0]
L(2) : [-0.5, 0.0, 1.5]
L(3) : [0.0, -1.5, 0.0, 2.5]
L(4) : [0.375, 0.0, -3.75, 0.0, 4.375]
L(5) : [0.0, 1.875, 0.0, -8.75, 0.0, 7.875]
```

the companion matrix

The *companion matrix* of $p = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5$ is

$$C_p = \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & -c_0/c_5 \\ 1 & 0 & 0 & 0 & -c_1/c_5 \\ 0 & 1 & 0 & 0 & -c_2/c_5 \\ 0 & 0 & 1 & 0 & -c_3/c_5 \\ 0 & 0 & 0 & 1 & -c_4/c_5 \end{array} \right].$$

The eigenvalues of C_p are the roots of p.

We apply `eigvals` of the `LinearAlgebra` module.

making the companion matrix in Julia

```
"""
    rootsCompanion(cff::Array{Float64,1})

returns the roots of the polynomial with coefficients cff,
by computing the eigenvalues of the companion matrix.
The last coefficient should not be zero.
"""
function rootsCompanion(cff::Array{Float64,1})
    lead = cff[end] # leading coefficient
    dim = length(cff) - 1
    companion = zeros(dim, dim)
    for k = 1:dim-1
        companion[k+1, k] = 1
    end
    for k = 1:dim
        companion[k, dim] = -cff[k]/lead
    end
    return eigvals(companion)
end
```

computing the roots of $L_5(x)$

```
L5 = [0.0, 1.875, 0.0, -8.75, 0.0, 7.875]
rootsL5 = rootsCompanion(L5)
for i=1:5
    sroot = @sprintf("%23.16e", rootsL5[i])
    value = evalpoly(rootsL5[i], L5)
    sterr = @sprintf("%.2e", value)
    println("r[", i, "] : $sroot : $sterr")
end
```

```
r[1] : -9.0617984593866252e-01 : 1.01e-14
r[2] : -5.3846931010568388e-01 : 1.91e-15
r[3] :  0.0000000000000000e+00 : 0.00e+00
r[4] :  5.3846931010568311e-01 : -1.20e-16
r[5] :  9.0617984593866596e-01 : 1.35e-14
```

roots of the Chebyshev polynomials

Exercise 3:

Chebyshev polynomials can be computed via the recursion:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x).$$

1. Define a Julia function `chebychev` which takes on input a degree d and which returns T_d as a `SymPy` expression.
   Your function should use a simple loop as in `legendre`.
2. Compute the roots of T_5 and verify the results using

$$x_i = \cos\left(\frac{(2i-1)\pi}{2n}\right), \quad i = 1, 2, \ldots, n,$$

   the theorem of lecture 16.

backward error using the 3-terms recursion

Exercise 4:

Chebyshev polynomials can be computed via the recursion:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$$

and have the roots

$$x_i = \cos\left(\frac{(2i-1)\pi}{2n}\right), \quad i = 1, 2, \ldots, n.$$

1. Use your function `chebyshev` of Exercise 3 to evaluate T_{100} at the roots x_i. Report the residuals $y_i = |T_{100}(x_i)|$.
2. Use the recursion for $T_{100}(x)$ to compute $z_i = |T_{100}(x_i)|$.

Compare the values y_i and z_i. Write a conclusion.

computation of the weights

The weights are in the solution vector of a linear system, constructed from the requirements that all polynomials to degree $2n - 1$ are integrated exactly.

$$\sum_{i=1}^{n} w_i x_i^d = \int_{-1}^{+1} x^d dx = \frac{(+1)^{d+1} - (-1)^{d+1}}{d+1}, \quad d = 0, 1, \ldots, 2n-1.$$

Instead of solving a linear system, we integrate the Lagrange polynomials:

$$w_i = \int_{-1}^{+1} \ell_i(x) dx, \quad \ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right),$$

where x_i are the points of the quadrature formula.

Gauss-Legendre quadrature with 5 points

```
$ julia gausslegendre.jl
L(5) = 63*x^5/8 - 35*x^3/4 + 15*x/8
Numeric coefficients :
L(5) : [0.0, 1.875, 0.0, -8.75, 0.0, 7.875]
The points :
r[1] : -9.061798459386252e-01 : 9.97e-15
r[2] : -5.384693101056838e-01 : 1.96e-15
r[3] :  0.000000000000000e+00 : 0.00e+00
r[4] :  5.384693101056831e-01 : 9.83e-18
r[5] :  9.061798459386596e-01 : 1.35e-14
The weights :
w[1] :  2.369268850561900e-01
w[2] :  4.786286704993645e-01
w[3] :  5.688888888888896e-01
w[4] :  4.786286704993685e-01
w[5] :  2.369268850561871e-01
$
```

degree of precision

A Gauss-Legendre quadrature with n points will integrate every polynomial of degree $2n - 1$ or less correctly.

Exercise 5:
Apply the five points and weights of the Gauss-Legendre to a random polynomial of degree nine and verify that the numerical approximation corresponds to the exact value computed with `SymPy`.

Exercise 6:
Use the five point Gauss-Legendre rule to demonstrate that the first ten Legendre polynomials form an orthogonal basis:

$$\langle L_i, L_j \rangle = \int_{-1}^{+1} L_i(x) L_j(x) dx$$

equals zero for all $j \neq i$ and one if $j = i$.

Gaussian Quadrature

reduction to an eigenvalue problem

If p_n is an orthogonal polynomial of degree n,
with the three terms recursion denoted as

$$p_{-1}(x) = 0, \ \ p_0(x) = 1, \ \ \text{for } j > 1 : p_j(x) = (a_j x + b_j) p_{j-1}(x) - c_j p_{j-2}(x),$$

then the roots of p_n are the eigenvalues of

$$J = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix} \quad \begin{array}{l} \alpha_i = -\dfrac{b_i}{a_i}, \\ \beta_i = \sqrt{\dfrac{c_{i+1}}{a_i a_{i+1}}}, \end{array}$$

$i = 1, 2, \ldots, n-1.$

weights of a Gauss quadrature rule

If $\mathbf{q}$ is the first row of Q,
of the orthogonal matrix with the eigenvectors of J in its columns,
then

$$w_i = q_i^2 \times \int_a^b w(x) dx$$

is the weight of the i-th point in the Gauss quadrature rule
with weight function $w(x)$ over the interval $[a, b]$, as in

$$\int_a^b w(x) f(x) dx \approx \sum_{i=1}^{n} w_i f(x_i), \quad \text{with } p_n(x_i) = 0, \ i = 1, 2, \ldots, n.$$

Main point: This construction scales well to make rules
with several hundreds of points.

an application: improper integrals

The integrand $f(x)$ of an improper integral $\displaystyle\int_a^b f(x)dx$ is undefined at some $x \in [a, b]$.

Example:

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} dx = \pi.$$

The weight of Gauss-Chebyshev quadrature is $w(x) = \dfrac{1}{\sqrt{1-x^2}}$.

Exercise 7:
Use the posted Jupyter notebook to apply a Gauss-Chebyshev quadrature rule with five points to $\displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} dx$.
What is the accuracy of your computation?

four lectures on differentiation and integration

1. Richardson extrapolation improves the accuracy of differences.
2. Quadrature rules are weighted sums of function evaluations and the weights are integrals of Lagrange polynomials.
3. By extrapolation, Romberg integration improves the accuracy of the composite trapezoidal rule.
4. Gaussian quadrature interpolates at the n roots of an orthogonal polynomial to reach a degree of precision equal to $2n - 1$.