MA378-Lab3

March 20, 2024

Table of Contents

1 2223-MA378 : Lab 3

1.1 Lab 3 of Numerical Analysis 2: Gaussian Quadrature

- Connect to [https://cloudjupyter.universityofgalway.ie/] and log in with the details provided to you in an email from `maths-stomaths-sto@universityofgalway.ie`
- Upload the notebook from https://www.niallmadden.ie/2324-MA378/#lab3
- Complete the tasks described below.
- Save and Export your notebook as a `.pdf` document;
- Upload that pdf to Canvas (2324-MA378… Assaignments… Lab 3) by 5pm, Friday 22nd March.

1.2 Collaboration policy

Collaboration is encouraged. If you work with some class-mates, please list their names in the first cell. However, every member of the team has to upload their own version.

[ ]:
```
1+1; # Just getting the gnuplot warning out of the way
```

1.3 Test Problem Data

In this lab, we'll take $f(x) = \log(2+x)$ and $[a,b] = [-1,1]$. The correct answer is

$$\int_{-1}^{1} \log(2+x)dx = 3\log(3) - 2.$$

Note: You'll need to copy this cell to a new notebook later.

[ ]:
```
a=-1; b=1;
f = @(x)log(2+x);
TrueInt = 3*log(3)-2;
```

1.4 Useful Functions

There are theee functions that we'll need for the rest of this notebook. You'll also need to copy them to your notebook.

1.4.1 `PolyDefInt()`

Because we'll often have to compute definite integrals of polynomials, to make the code more readable, we'll define a function `PolyDefInt(p,a,b)`, that returns $\int_a^b p(x)dx$.

[ ]:
```
function int_ans = PolyDefInt(p,a,b)
g = polyint(p);
int_ans = polyval(g,b)-polyval(g,a);
```

```
end
```

1.4.2 `LagrangePoly()`

Often, we'll need to compute the Lagrange Polynomails for a set of points. This function can do that. It returns the `polyval()` representation of $L_i(x)$ for a given set of points `x` and index `i`.

[ ]:
```
function Li = LagrangePoly(x,i)
n = length(x)-1;
Li = 1; % will use conv() to build the Lagrange polynomial
for k=1:n+1
   if (k ~= i)
      Li = conv(Li, [1,-x(k)])/(x(i)-x(k));
   end
end
end
```

1.4.3 `IP()`

To use the algorithm, we need to complete numerous inner products. Do simplify the code, here is a function called `IP()` that computes $IP(p,q) = \int_{-1}^{1} p(x)q(x)dx$ where `p` and `q` are vectors representing polynomials.

[ ]:
```
function v = IP(p,q)
r = conv(p,q);
v = PolyDefInt(r,-1,1);
end
```

1.5 Numerical Integration

Octave has some built-in functions for numerical integration, but we'll are interested in writing our own.

Given a function $f(x)$, we want to approximate $\int_a^b f(x)dx$ We've learned about two ways to do this: **Newton-Cotes Quadrature** methods, and **Gaussian Quadrature**.

1.6 Newton-Cotes methods

For a Newton-Cotes method, $Q_N(\cdot)$, we set $x = \{x_0, x_1, \ldots, x_N\}$, to be a set of $N+1$ equally spaced points on $[a,b]$, including the two end points. These are called the *quadrature points*. We then let $q = \{q_0, q_1, \ldots, q_N\}$ be a set of *quadrature wieghts* so that $Q_N(p) = \int_a^b p(x)dx$, for any polynomial, pm of degree N or less. (Review notes from Section 3.2 to see how these points are calculated). The method is then

$$Q_N(f) := \sum_{i=0}^{N} q_i f(x_i).$$

1.6.1 Trapezium Rule

For the Trapexium Rule, denoted, $Q_1(\cdot)$, we have

```
[ ]: x = [0, 1]; % Quadrature points
     q = [1, 1]; % Quadrature weights
```

We could compute this as `q(1)*f(x(1)) + q(2)*f(x(2))`. However, the following code works for any pair of vectors `q` and `x`, so long as they are the same length.

```
[ ]: Q1 = sum(q.*f(x));
     Q1Error = abs(TrueInt - Q1);
     fprintf('Q1: Estimate = %8.6f, Error = %8.3e\n', Q1, Q1Error);
```

1.6.2 Simpson's Rule

```
[ ]: n=2;
     x = linspace(a,b,n+1);
     q = [1,4,1]/3;
     Q2 = sum(q.*f(x));
     Q2Error = abs(TrueInt - Q2);
     fprintf('Q%d: Estimate = %8.6f, Error = %8.3e\n', n, Q2, Q2Error);
```

1.6.3 Four point Newton-Cotes

The 4-point scheme has weights: $(\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4})$. Here is its estimate

```
[ ]: n=3;
     x = linspace(a,b,n+1);
     q = [1,3,3,1]/4;
     Q3 = sum(q.*f(x));
     Q3Error = abs(TrueInt - Q3);
     fprintf('Q%d: Estimate = %8.6f, Error = %8.3e\n', n, Q3, Q3Error);
```

There is also a 5-point rule called *Boole's Rule* (named after Cork's George Boole). You can read about it here: https://en.wikipedia.org/wiki/Boole's_rule

1.6.4 Deriving n-point Newton-Cotes

In general, a Newton-Cotes method may be constructed by first choosing the points and then computing $q_k = \int_a^b L_k(x)dx$, where $L_k(x)$ is the Lagrange Polynomial associated with the point x_k. In the next section, we'll see how to compute that.

1.7 Polynomials in Octave

In Ocatve and MATLAB, a polynomial is represented as a vector: * If the vector is called `p`, then the degree of the polynomial is `n=length(p)-1`. * `p(1)` is the coefficent of x^n, `p(2)` is the coefficient of x^{n-1}, ..., `p(n+1)` is the coefficient of $x^0 = 1$. In general, `p(k)` is the coefficient of x^{n-k+1}.

1.7.1 Examples:

- If `p` is the vector representing a monic polynomial, then 'p(1)=".
- The vector for $x^2 - 1$ is `p=[1,0,-1]`.
- `p=[5,-4,3,-2]` is the vector for $5x^3 - 4x^2 + 3x - 2$.

1.7.2 Functions

Octave/MATLAB has several useful functions for working with polynomials. Here are some we'll use in this lab.

`polyval()` Given vectors `p` and `x`, the function call `polyval(p,x)` returns the values of the polynomial represented by `p` at the points in `x`.

Example: let's check that $x^2 + x - 6 = (x-2)(x+3)$ is zero at $x = 2$ and $x = -3$.

```
[ ]: p2=[1,1,-6]
     polyval(p2,[-3, 2])
```

Plot the interpolant and the error

`polyval()` is particularly useful for plotting. To plot a polynomial represented by the vector `p` on the interval $[a, b]$, do something like this:

```
[ ]: p3=[1,2,-5,-6]; % reps x^3+2x^2-5x-6
     a=-4; b=3; % change these to anything you like.
     x=linspace(a,b,100); % vector with 100 points in [a,b]
     plot(x, polyval(p3,x), x,x*0, ':')
```

`roots()` Given a vector `p` that represents a polynomial, the `roots()` function returns its roots. In the previous example, the plot suggests `p3` has roots at $x = -3$, $x = -1$, and $x = 2$. Let's check:

```
[ ]: roots(p3)
```

`conv()` The `conv()` function can be used to multiply polynomials. The name comes from an operation on vectors called convolution, which is equivalent to polynomial interpolation. To try it out, let's recall that the ploynomial $p_1(x) = x - s$ is the polynomial of degree 1 with a zero at $x = s$. In MATLAB, we could represent it as p1=[1,-s]. In an earlier example we had a polynomial $p_2 = x^2 + x - 6 = (x-2)(x+3)$. To make this using `conv()`:

```
[ ]: p2=conv([1,2],[1,-3])
```

`polyint()` The `polyint()` function returns the coefficients for the polynomial that is the antidervative of the given one, assuming a constant of integration of 0.

For example, we know that $\int_a^b x^3 + 2x^2 - 5x - 6dx = \frac{1}{4}x^4 + \frac{2}{3}x^3 - \frac{5}{2}x^2 - 6x + C$. Let's check

```
[ ]: g=polyint(p3)
```

In numerical integration, we are usually interested in computing definite integrals. To compute $\int_{-1}^{1} p_3(x)dx$, first set g to be the antiderivative of p_3, and then use `polyval()` to compute $g(-1)$ and $g(1)$. Subtract the first from the second to get the desired value.

```
[ ]: polyval(polyint(p3),1)-polyval(polyint(p3),-1)
```

Check that the `PolyDefInt()` function above works:

```
[ ]: PolyDefInt(p3,-1,1)
```

1.8 Deriving Newton-Cotes Rules

We now have all the tools needed to compute quadrature weights for Newton-Cotes methods. As an example, let's reproduce that -point scheme.

```
[ ]: n = 3;
     a = -1; b = 1;
     x = linspace(a,b,n+1);
     q = zeros(1,n+1); % vector where we will store the weights
```

Next we define the weights as inetgrals of the Lagrange polynomials:

$$L_i = \prod_{k=0,k\neq 1}^{n} \frac{x - x_k}{x_i - x_k}.$$

In the following, take care to note that Octave indexes from 1, and not 0.

```
[ ]: for i=1:n+1
        Li = 1; % will use conv() to build the Lagrange polynomial
        for k=1:n+1
           if (k ~= i)
              Li = conv(Li, [1,-x(k)])/(x(i)-x(k));
           end
        end
        q(i) = PolyDefInt(Li,a,b);
     end
     disp(q) % output values of q
```

You should check that these are the same values as use above. Sometimes, it can be helpful to look at rational expressions for these numbers:

```
[ ]: rat(q)
```

1.9 Gaussian Quadrature

The Gaussian Quadrature method, $G_n(\cdot)$, is based on choosing the weights and the points to optimise the precision of the method. We've learned in class the that trick to computing these values involves working with sequences of orthogonal monic polynomials.

1.9.1 Sequences of monic orthogonal polynomials

Review the notes from Section 3.5, and especially Theorem 5.11. See https://www.niallmadden.ie/2324-MA378/3-5-OrthogonalPolynomials.pdf. Because we are using Octave, we have to change the notation: we'll represent $\tilde{p}_k$ `p{k+1}`, since Octave indexes from zero.

Start by setting $\tilde{p}_1 = 1$.

```
[ ]: p{1} = 1; % this is what we used call p0. But Octave indexes from 1,␣
     ↪unfortunately.
```

The we compute $\tilde{p}_2 = x - \alpha$.

```
[ ]: alpha = IP( conv([1,0], p{1}),p{1})/IP(p{1},p{1}); % Note [1,0] is the␣
     ↪polynomial 'x'.
     p{2} = [1,-alpha];
```

After that, we use $\tilde{p}_{n+1}(x) = (x - \alpha)\tilde{p}_n(x) - \beta\tilde{p}_{n-1}(x)$. Here is an example. Note that in the last line, we padded `p{n-1}` with 2 zeros, because we can only add vectors of the same degree/length.

```
[ ]: n = 2;
     alpha = IP( conv([1,0], p{n}),p{n})/IP(p{n},p{n});
     beta  = IP( conv([1,0], p{n}),p{n-1})/IP(p{n-1},p{n-1});
     p{n+1} = conv([1,-alpha], p{n}) - beta*[0,0,p{n-1}];
```

From Example 5.13 of Section 3.5 (see https://www.niallmadden.ie/2324-MA378/3-4-GaussianQuadrature.pdf) we know that, with $a = -1$ and $b = 1$, we should get the first three polynomials are 1, x and $x^2 - 1/3$. Furthermore, the roots of $x^2 - 1/3$ are the quadrature points for $G_1(x)$, i.e.,

$$x_0 = -\frac{1}{\sqrt{3}} \quad \text{and } x_1 = \frac{1}{\sqrt{3}}.$$

Let's check:

```
[ ]: x2=roots(p{3})'  % transposed to row vector
```

```
[ ]: x2.^2
```

1.9.2 Gaussian Quadrature Weights

The weights are computed in the same was as for Newton-Cotes: given a set of points, compute the Lagrange polynomials for each of these. Their integrals on $[a, b]$ are the weight.

[ ]:
```
n = 2;
x = roots(p{n+1})'; % make sure that p{n+1} has been computed above.
w = zeros(1,n); %
for i=1:n
   w(i) = PolyDefInt(LagrangePoly(x, i),-1,1);
end
disp(w)
```

Now we can apply the method. The code is essentially the same as for Newton-Cotes.

[ ]:
```
G1 = sum(w.*f(x));
G1Error = abs(TrueInt - G1);
fprintf('G1: Estimate = %8.6f, Error = %8.3e\n', G1, G1Error);
```

1.10 Your Tasks

1. Create a new (blank) Jupuyer Notebook, which uses the `Calysto Octave`. Add your name, ID number and email address in a `markdown` cell at the top. also name anyone you collaborated with. (If you prefer, you can delete ALL the unneeded stuff from this notebook, but it is probably easier to start a new notebook, and just copy over what is needed).
2. Copy the test problem data (values of `a`, `b`, `f` and `TrueInt`) from the `TestProblemData` section above.
3. Copy the functions from the `Useful Functions` section above to your notebook.
4. Write a piece of code that derives and implements the 5-point **Newton-Cotes method**, and tests it when estimating
$$\int_a^b f(x)dx = \int_{-1}^1 \log(x+2)dx.$$
The tests should include reporting the error.
5. Write a piece of code that derives and implements the 3-point and 4-point **Gaussian** methods, and tests them when estimating $\int_{-1}^1 \log(x+2)dx$. The tests should include reporting the error.
6. Export your notebook as a PDF document, and upload that to Canvas (2324-MA378 - Assignments - Lab 3). **Deadline: 5pm, Friday 22 March.**