University of Costa Rica
School of Mathematics
MA-501 Numerical Analysis
Homework 2

Juan Ignacio Padilla B.
ID: B55272
Prof. Juan Gabriel Calvo
January 29, 2026

Homework 2

Problem 1.

(Evaluating polynomials) Consider a polynomial of degree n written in the standard basis

$$p(x) = \sum_{k=0}^{n} a_k x^k,$$

and define the coefficient vector

$$\boldsymbol{a} = [a_0, \cdots, a_n]^T.$$

For a given value x_0, we want to evaluate $p(x_0)$. To do this, we consider 3 different methods:

Alg. 1) By direct evaluation, we compute each term $a_k x^k$ and then sum all the terms.

Alg. 2) Define the sequence $\{b_k\}_{k=0}^n$ by backward recursion

$$\begin{aligned} b_n &= a_n, \\ b_k &= a_k + b_{k+1}x_0 \quad \text{for } k = n-1, n-2, \cdots, 0. \end{aligned}$$

Then $p(x_0) = b_0$

Alg. 3) Use the `polyval()` command in `MATLAB` to evaluate the polynomial.

(a) Determine the number of operations (additions and multiplications) that must be performed in Algorithm 1. We have:

$$\begin{aligned} a_0 &\to 0 \text{ operations.} \\ a_1 x &\to 1 \text{ operation.} \\ a_2 x^2 &\to 2 \text{ operations.} \\ &\vdots \\ a_n x^n &\to n \text{ operations.} \end{aligned}$$

So in total we have to perform $n(n+1)/2$ multiplications. Taking into account the n additions that must be performed, we get a total of

$$\frac{n^2 + 3n}{2}$$

operations.

(b) Determine the number of operations (additions and multiplications) that must be performed in Algorithm 2.

Observe that in each iteration $b_k = a_k + b_{k+1}x_0$, 2 operations are needed (except in the first one). Multiplying by the number of iterations (n, since from 0 to $n-1$ there are n numbers), we have that $2n$ operations are needed to compute b_0.

(c) For Algorithm 2, prove that $p(x_0) = b_0$.

We will proceed by induction on the degree of p. For $n = 0$, the situation is trivial since if $p(x) = a_0$, the first iteration $b_0 = a_0$ satisfies $p(x_0) = b_0$. Assume the validity of the result for polynomials of degree $n-1$ and let

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0.$$

Consider the polynomial

$$q(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_1.$$

Applying the recursive formula for $q(x)$

$$\begin{aligned} c_n &= a_n, \\ c_k &= a_k + c_{k+1}x_0 \quad \text{for } k = n-1, n-2, \ldots, 1. \end{aligned}$$

We obtain by the induction hypothesis that $c_1 = q(x_0)$. Now, taking the recursion for $p(x)$,

$$\begin{aligned} b_n &= a_n, \\ b_k &= a_k + b_{k+1}x_0 \quad \text{for } k = n-1, n-2, \ldots, 0. \end{aligned}$$

We see that $b_k = c_k$ for $k = 1, \ldots, n$, since the recursions coincide. It remains to see what happens when $k = 0$. We have

$$\begin{aligned} b_0 &= a_0 + b_1 x_0 \\ &= a_0 + c_1 x_0 \\ &= a_0 + q(x_0)x_0 \\ &= a_0 + x_0(a_n x_0^{n-1} + a_{n-1}x_0^{n-2} + \cdots + a_1) \\ &= p(x_0). \end{aligned}$$

(d) Compare the execution time of Algorithms 1, 2 and 3 in `MATLAB`. To do this, generate a vector with random entries from a uniform distribution on $(0, 1)$ for the coefficients using the command

```
a=rand(n,1)
```

Use $n = 10^8$, $x_0 = 0.1$ and compare the execution times.

Using `MATLAB`, when applying algorithm 1 to a random vector of size 10^8, which generates a polynomial of degree $10^8 - 1$, **an average time of 11.6280274 seconds was obtained** (the test was performed 5 times).

For the case of algorithm 2, the operations took **an average of 0.2776524 seconds** (in 5 tests). This represents an enormous improvement compared to algorithm 1, which may be because the number of operations performed is linear with respect to n, while in algorithm 1, it is of quadratic order.

Finally, for algorithm 3, **it took an average of 1.2224204 seconds** (in 5 tests as well). It would seem that, in this situation, Algorithm 2 is the fastest. However, this cannot be assumed in general, since our coefficients are all positive and less than 1. Perhaps with a different nature of coefficients, different results would be obtained.

Note: Since we have random polynomials, the value of $p(x_0)$ is not specified; however, in the attached script it can be verified that indeed the values coincide for all 3 algorithms.

(e) Now consider `a= ones(n,1)`, $n = 10^8$, $x_0 = 0.1$. What is the exact value of $p(x_0)$? Use algorithm 1 and 2 to obtain approximations $\tilde{p}_1$ and $\tilde{p}_2$, respectively. How many significant figures are correct for $\tilde{p}_1$ and $\tilde{p}_2$?

Let us calculate the exact value of $p(x_0)$. We have

$$p(x) = \sum_{n=0}^{10^8-1} x^n.$$

Then

$$p(0.1) = \sum_{n=0}^{10^8-1} \left(\frac{1}{10}\right)^n$$

Which can be calculated using the geometric formula to obtain

$$p(0.1) = \left(\frac{1 - \left(\frac{1}{10}\right)^{10^8}}{1 - \frac{1}{10}}\right) = \frac{10^{10^8} - 1}{9 \times 10^{10^8 - 1}}$$

Or more easily we can see that

$$\begin{aligned} p(0.1) &= 1 + 0.1 + 0.001 + \cdots + \underbrace{0.0\cdots01}_{10^{10^8} \text{ zeros}} \\ &= \underbrace{1.1\cdots1}_{10^{10^8} \text{ ones}} \end{aligned}$$

The result of Algorithm 1 is $\tilde{p}_1 = 1.111111111111111$, so it is exact with 16 significant figures. While the result of applying Algorithm 2 is also $\tilde{p}_2 = 1.111111111111111$, which is also exact with 16 significant figures. We conclude therefore that both algorithms have high accuracy. For obvious reasons, it is not logical to expect that `MATLAB` can output the 100 million decimals that the exact value of $p(x_0)$ has; however, the approximations given by the algorithms are reasonably close.

Problem 2.

Suppose now that we perform a change of basis and express the polynomial in the Chebyshev polynomial basis,

$$p(x) = \sum_{k=0}^{n} c_k T_k(x), \quad |x| \leq 1,$$

Where $T_k(x) = \cos(k \arccos x)$. In class we proved the recurrence relation

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)$$

We want to compute $p(x_0)$. To do this, consider the following algorithm:

Alg 4) Define the sequence $\{b_k(x_0)\}_{k=0}^{n+2}$ by backward recurrence

$$\begin{aligned} b_{n+2}(x_0) &= b_{n+1}(x_0) = 0, \\ b_k(x_0) &= c_k + 2x_0 b_{k+1}(x_0) - b_{k+2}(x_0), \quad \text{for } k = n, n-1, \ldots, 1 \end{aligned} \tag{$\star$}$$

The desired value is then

$$p(x_0) = T_0(x_0)c_0 + T_1(x_0)b_1(x_0) - T_0(x_0)b_2(x_0).$$

a) Determine the number of operations that must be performed in Algorithm 4.

We have 4 operations in each iteration. Having exactly n iterations, there will be a total of $4n$ operations.

b) Verify that formula $(\star)$ is correct.

First note that $T_0(x) = 1$, and $T_1(x) = x$. Similarly, we proceed by strong induction on n. If $p(x) = c_0 T_0 = c_0$, then the recursion simply takes $b_2 = b_1 = 0$ and

$$T_0(x_0)c_0 + T_1(x_0)b_1(x_0) - T_0(x_0)b_2(x_0) = c_0 + x_0 b_1 - b_2 = c_0 = p(x_0).$$

Now assume the result for linear combinations of the first k Chebyshev polynomials, with $k < n$. Let

$$p(x) = \sum_{k=0}^{n} c_k T_k(x),$$

and define its recursion:

$$\begin{aligned} b_{n+2} &= b_{n+1} = 0, \\ b_k(x_0) &= c_k + 2x_0 b_{k+1} - b_{k+2} \quad \text{for } k = n, n-1, \ldots, 1 \end{aligned}$$

Now consider the polynomials

$$q(x) = \sum_{k=0}^{n-1} c_{k+1} T_k(x) \qquad ; \qquad r(x) = \sum_{k=0}^{n-2} c_{k+2} T_k(x)$$

For q, we define the recurrence

$$\begin{aligned} d_{n+1} &= d_n = 0, \\ d_k &= c_k + 1 + 2x_0 d_{k+1} - d_{k+2} \quad \text{for } k = n-1, \ldots, 1 \end{aligned}$$

We also define the recurrence for r:

$$\begin{aligned} u_{n-1} &= u_n = 0, \\ u_k &= c_k + 2 + 2x_0 u_{k+1} - u_{k+2} \quad \text{for } k = n-2, \ldots, 1 \end{aligned}$$

We have, by the induction hypothesis, that

$$q(x_0) = c_1 + x_0 d_1 - d_2$$

$$r(x_0) = c_2 + x_0 u_1 - u_2$$

Also observe that the first steps of the recursions for p, q and r coincide. More specifically, $b_{k+1} = d_k$ for $k = 1, \ldots, n+1$ and $b_{k+2} = u_k$ for $k = 1, \ldots, n$. This allows us to observe that, returning to the recursion for p:

$$\begin{aligned} b_1 &= c_1 + 2x_0 b_2 - b_3 \\ &= c_1 + 2x_0 d_1 - d_2 \\ &= 2q(x_0) - c_1 + d_2 \end{aligned}$$

And also that

$$\begin{aligned} b_2 &= c_2 + 2x_0 b_3 - b_4 \\ &= c_2 + 2x_0 u_1 - u_2 \\ &= r(x_0) + x_0 u_1 \end{aligned}$$

We finally have

$$
\begin{aligned}
&T_0(x_0)c_0 + T_1(x_0)b_1(x_0) - T_0(x_0)b_2(x_0) \\
&= c_0 + x_0 b_1 - b_2 \\
&= c_0 + 2x_0 q(x_0) - c_1 x_0 + \cancel{x_0 d_2} - r(x_0) - \cancel{x_0 u_1} \qquad \text{since } u_1 = d_2 \\
&= c_0 - c_1 x_0 + 2x_0 \sum_{k=0}^{n-1} c_{k+1} T_k(x_0) - \sum_{k=0}^{n-2} c_{k+2} T_k(x_0) \\
&= c_0 - c_1 x_0 + 2x_0 \sum_{k=1}^{n} c_k T_{k-1}(x_0) - \sum_{k=2}^{n} c_k T_{k-2}(x_0) \\
&= c_0 - c_1 x_0 + 2x_0 c_1 T_0(x_0) + \sum_{k=2}^{n} c_k (2x_0 T_{k-1}(x_0) - T_{k-2}(x_0)) \\
&= c_0 + c_1 x_0 + \sum_{k=2}^{n} c_k T_k(x_0) \\
&= p(x_0).
\end{aligned}
$$

c) Write a function `y0=evalCheb(c,x0)` in `MATLAB` that computes $y_0 = p(x_0)$ using formula $(\star)$, where the input is the coefficient vector $\mathbf{c}$ and the value x_0.

The function `y0=evalCheb(c,x0)` was implemented, attached in the files.

d) Use `c=rand(n,1)`, with $n = 10^8$, $x_0 = 0.1$ and compare the execution time with question (1d). Comment.

When applying the function `y0=evalCheb(c,x0)` to 5 random coefficient vectors between 0 and 1, of size 100 million, an **average evaluation time of 0.3781628 seconds** was obtained, which is substantially faster than algorithms 1 and 3. However, it is slightly slower than algorithm 2. All of this can be justified again by looking at the number of operations each algorithm needs. If we recall that 1 was of quadratic order with respect to n, it is no surprise that it is the slowest, since 3 and 4 are of linear order, with 4 being a bit slower since it needs twice as many operations as 2. Once again, without knowing exactly what operations `polyval()` uses, a discussion of this type is not possible; we can only say that it has an intermediate evaluation time compared to the others.

e) Consider the polynomial $p(x) = x^3 - 3x^2 + 1$. Express $p(x)$ as a linear combination of Chebyshev polynomials

$$p(x) = c_0 T_0(x) + c_1 T_1(x) + c_2 T_2(x) + c_3 T_3(x)$$

and verify the behavior of Algorithm 4 for $x_0 = 1$.

We have that $T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1$ and $T_3(x) = 4x^3 - 3x$. Then we just need to solve the system of equations in 4 variables $\alpha, \beta, \gamma, \delta$

$$x^3 - 3x^2 + 1 = \alpha(4x^3 - 3x) + \beta(2x^2 - 1) + \gamma x + \delta$$

Where it is easily seen that $\alpha = 1/4, \beta = -3/2, \gamma = 3/4$ and $\delta = -1/2$. When applying Algorithm 4 in the Chebyshev basis, we obtain the result $p(1) = -1$, which verifies the functioning of the algorithm numerically.

Problem 3

(Integrating Chebyshev expansions) With the notation of the previous exercise, consider the polynomial

$$p_n(x) = \sum_{k=0}^{n} c_k T_k(x), \qquad |x| \leq 1$$

a) Prove that for every integer $n \in \mathbb{N}$, $n \neq 1$,

$$\int_{-1}^{1} T_n(x)dx = \frac{1+(-1)^n}{1-n^2}.$$

Deduce that

$$\int_{-1}^{1} p_n(x)dx = \sum_{\substack{k=0 \\ k \text{ even}}} \frac{2c_k}{1-k^2}$$

Solution: If $n = 0$, the result is trivial, since $T_0 = 1$. For $n > 1$, consider

$$T_{n+1}(x) = \cos((n+1)\arccos(x))$$

We differentiate to obtain

$$\frac{T'_{n+1}(x)}{n+1} = \frac{\sin((n+1)\arccos(x))}{\sqrt{1-x^2}}.$$

Similarly for $T_{n-1}(x)$, we have

$$\frac{T'_{n-1}(x)}{n-1} = \frac{\sin((n-1)\arccos(x))}{\sqrt{1-x^2}}.$$

Now let $\theta = \arccos(x)$. Subtracting the previous expressions, and applying some trigonometric identities, we obtain

$$\begin{aligned}\frac{T'_{n+1}(x)}{n+1} - \frac{T'_{n-1}(x)}{n-1} &= \frac{\sin((n+1)\theta) - \sin((n-1)\theta)}{\sin(\theta)} \\ &= \frac{2\cos(n\theta)\sin(\theta)}{\sin(\theta)} \\ &= 2T_n(x).\end{aligned}$$

We integrate this identity

$$\int_{-1}^{1} T_n(x) = \frac{1}{2}\int_{-1}^{1} \frac{T'_{n+1}(x)}{n+1} - \frac{T'_{n-1}(x)}{n-1}dx$$

$$\int_{-1}^{1} T_n(x) = \frac{1}{2(n+1)}T_{n+1}(x)\Big|_{-1}^{1} - \frac{1}{2(n-1)}T_{n-1}(x)\Big|_{-1}^{1}$$

Now, note that for all n,

$$T_n(1) = 1$$

$$T_n(-1) = \cos(n\pi) = (-1)^n$$

Therefore

$$\int_{-1}^{1} T_n(x) = \frac{1-(-1)^{n+1}}{2(n+1)} - \frac{1-(-1)^{n-1}}{2(n-1)} = \frac{1+(-1)^n}{1-n^2}$$

Which is what was sought.[1] Now, for $p_n(x) = \sum_{k=0}^{n} c_k T_k(x)$ We simply have

$$\begin{aligned}\int_{-1}^{1} p_n(x)dx &= \sum_{k=0}^{n} c_k \int_{-1}^{1} T_k(x)dx \\ &= \sum_{k=0}^{n} c_k \frac{1+(-1)^k}{1-k^2} \\ &= \sum_{\substack{k=0 \\ k \text{ even}}}^{n} \frac{2c_k}{1-k^2}\end{aligned}$$

b) Program a function `I = intCheb(c)`, whose input is the coefficient vector

$$\mathbf{c} = [\mathbf{c_0}, \ldots, \mathbf{c_n}]^{\mathbf{T}}$$

given in (3), that computes the value I of integral (4).

The function `intCheb(c)` was implemented, which is attached in the files.

Problem 4.

Given $n \in \mathbb{N}$, consider the $n+1$ Chebyshev points given by

$$x_j = \cos\left(\frac{j\pi}{n}\right) \quad j = 0, 1, \ldots, n.$$

Given a function f it is possible to write the Lagrange interpolating polynomial with nodes $\{x_j\}_{j=0}^{n}$ in the Chebyshev polynomial basis

$$p_n(x) = \sum_{k=0}^{n} f(x_k) L_k(x) = \sum_{k=0}^{n} c_k T_k(x).$$

Knowing the values $\{f(x_k)\}_{j=0}^{n}$ we can efficiently compute the coefficients c_k using the inverse discrete Fourier transform, which is implemented with the `ifft` command in `MATLAB`. The following function receives as input the values $\{f(x_k)\}_{j=0}^{n}$ in a column vector and returns the coefficients in a vector $\mathbf{c}$.

```
function c= val2coef(values)
n=size(values,1);
% Mirror the values
tmp=[values(n:-1:2,:);values(1:n-1,:)];
% apply ifft
c=real(ifft(tmp));
%Truncate
c=c(1:n,:);
%Scale the interior coefficients
c(2:n-1,:)=2*c(2:n-1,:);
end
```

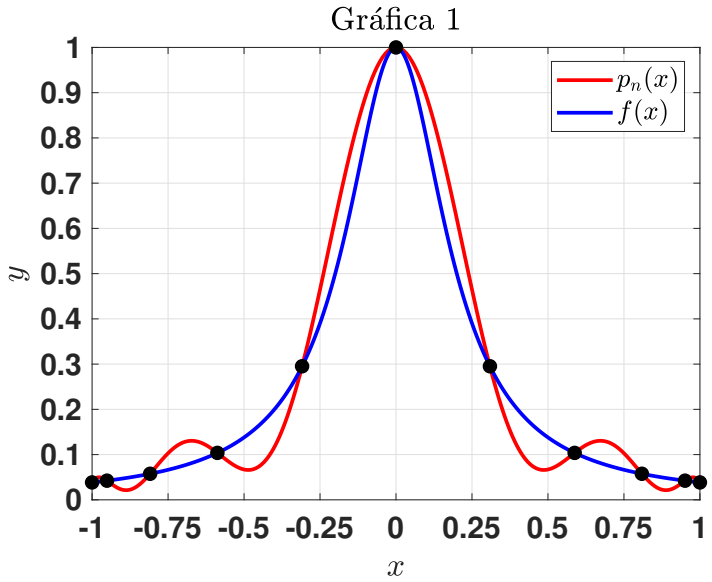
a) Implement a function `[coef,nodes] = interpCheb(n,f)` that receives the degree n and the function f, which computes the Chebyshev nodes and the coefficients $\{c_k\}$.

The function `[coef,nodes] = interpCheb(n,f)` was implemented, attached in the files.

[1] The omitted calculations are straightforward.

b) For $f(x) = 1/(1+25x^2)$, $n = 10$, compute the coefficients `c=interpCheb(n,f)`. Define `xx = linspace(-1,1,1e3)` and use the `evalCheb` command to evaluate p_n at each entry of `xx`. Plot f and p_n on the same graph, including the interpolation nodes. Comment on your result. Does the Runge phenomenon occur?

Following the instructions, we obtain figure 1.



For $n = 10$, the approximation is not too good, as there is a certain degree of fluctuation between the values of $p_n(x)$ and $f(x)$. We can see, however, that at the Chebyshev nodes (the black points), the curves do in fact coincide. We conclude that for $n = 10$, it would seem that the Runge phenomenon **has not completely disappeared.**

c) Plot $||f(\texttt{xx}) - p_n(\texttt{xx})||_{\ell\infty}$ as a function of n for $n \in \{2, \ldots, 20\}$. What rate of convergence do you observe as n increases?

When plotting the error, we obtain Graph 2:

It is observed that the error alternates, being slightly larger for odd n; this could be due to the symmetry of the function (when choosing asymmetric nodes, the interpolation values will in turn be asymmetric). However, regardless of the fluctuation, the error decreases as n increases, with a rate that seems to be **superlinear**.

d) For $f(x) = 1/(1+25x^2), n = 10$ use the formula from 3a) to approximate $\int_{-1}^{1} f(x)dx$. Plot the absolute error as a function of n for $n \in \{2, \ldots, 20\}$. What rate of convergence do you observe as n increases?

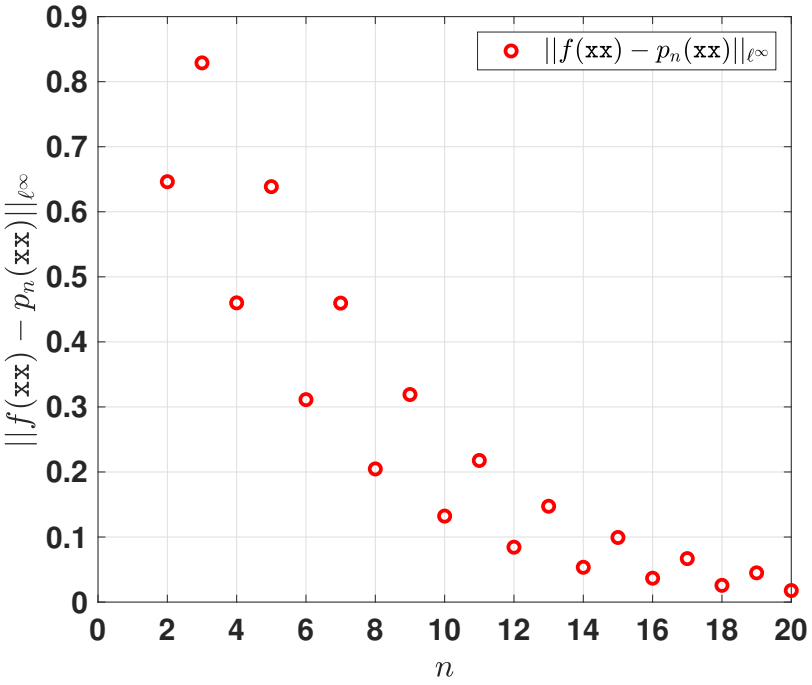
When applying the function `intCheb` to f with $n = 10$, we obtain that

$$\int_{-1}^{1} f(x)dx \approx 0.573232153266308$$

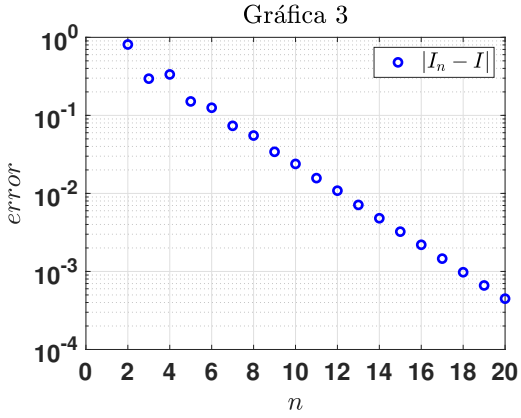
Which approximates the exact value of $\frac{2}{5}\arctan 5 \approx 0.549360306778006344$ with 1 decimal. For different n we have graph 3.

Where I_n is the approximation of the integral I, using polynomials in the Chebyshev basis, of dimension n.

Gráfica 2



Gráfica 3



It can be observed that the rate of convergence is almost perfectly **linear**, so integration by Chebyshev coefficients seems a good numerical option.

e) What advantages do you observe in interpolating at Chebyshev nodes with Chebyshev polynomial expansions to approximate integrals?

The main advantage of interpolating using Chebyshev nodes is the fact that the **Runge phenomenon disappears** for high values of n; this eliminates the fluctuations that may appear at the ends of the interpolation interval, which in turn improves the reliability of the integral. As a secondary advantage, there is the ease with which these integrals are computed. By expressing the polynomial in the Chebyshev basis (which does not represent major computational difficulties, since it is just a change of basis), we have that the integral of the interpolating polynomial is given by the formula

$$\int_{-1}^{1} p_n(x)dx = \sum_{\substack{k=0 \\ k \text{ even}}} \frac{2c_k}{1-k^2}$$

Which is simple to calculate, since it only involves the coordinates of the polynomial in the basis, and additions, multiplications, and divisions. We conclude therefore that Chebyshev polynomials represent a useful tool in numerical analysis, both in interpolation and in integration of smooth functions.

Problem 5.

Construct the quadrature formula

$$\int_a^b f(x)w(x)dx \approx W_0 f(a) + \sum_{k=1}^{n} W_k f(x_k),$$

where we have fixed the node $x_0 = a$; that is, calculate the weights $\{W_k\}_{k=0}^n$ and nodes $\{x_k\}$ so that the above formula is exact for polynomials of degree at most $2n$. Suggestion: Write

$$p_{2n}(x) = (x-a)q_{2n-1}(x) + r,$$

with $r \in \mathbb{R}$.
Following the suggestion, we will assume that f is a polynomial of degree $2n$ (it could be interpreted as some polynomial that interpolates it). Note that, evaluating at a, we immediately have $r = p_{2n}(a)$.
Consider the weight

$$\tilde{w}(x) = (x-a)w(x).$$

We can apply Gaussian quadrature to $q_{2n-1}(x)$ to have

$$\int_a^b \tilde{w}(x)q_{2n-1}(x)dx = \sum_{k=1}^{n} \tilde{W}_k q_{2n-1}(\tilde{x}_k)$$

Where $\{\tilde{x}_k\}_{k=1}^n$ are the nodes of Gaussian quadrature applied to the weight $\tilde{w}$, and $\{\tilde{W}_k\}_{k=1}^n$ their respective weights.
Since

$$p_{2n}(x) = (x-a)q_{2n-1}(x) + p_{2n}(a)$$

Multiplying by $w(x)$, and integrating we have

$$\begin{aligned}\int_a^b p_{2n}(x)w(x)dx &= \int_a^b \tilde{w}(x)q_{2n-1}(x)dx + \int_a^b p_{2n}(a)w(x)dx \\ &= \sum_{k=1}^{n} \tilde{W}_k q_{2n-1}(\tilde{x}_k) + p_{2n}(a)\int_a^b w(x)dx\end{aligned}$$

But since $q_{2n-1}(\tilde{x}_k) = \frac{p_{2n}(\tilde{x}_k) - p_{2n}(a)}{\tilde{x}_k - a}$, we have

$$\begin{aligned}\int_a^b p_{2n}(x)w(x)dx &= \sum_{k=1}^{n} \tilde{W}_k \frac{p_{2n}(\tilde{x}_k) - p_{2n}(a)}{\tilde{x}_k - a} + p_{2n}(a)\int_a^b w(x)dx \\ &= \sum_{k=1}^{n} \frac{\tilde{W}_k}{\tilde{x}_k - a} p_{2n}(\tilde{x}_k) - \sum_{k=1}^{n} \frac{\tilde{W}_k p_{2n}(a)}{\tilde{x}_k - a} + p_{2n}(a)\int_a^b w(x)dx\end{aligned}$$

Therefore we can take

$$W_0 = \int_a^b w(x)dx - \sum_{k=1}^{n} \frac{\tilde{W}_k}{\tilde{x}_k - a}$$

$$W_k = \frac{\tilde{W}_k}{\tilde{x}_k - a} \quad \text{for} \quad 1 \leq k \leq n$$

And taking $x_k = \tilde{x}_k, k = 1, \ldots, n$ we obtain the desired quadrature. Observe that $W_k > 0$, since $\tilde{W}_k > 0$. For $k = 0$, define $P(x) = (b - x) \prod_{k=1}^{n} (x - x_k)^2$; the quadrature we constructed immediately yields

$$0 < \int_a^b P(x)w(x)dx = W_0 P(a)$$

Therefore $W_0 > 0$. Finally, having defined n nodes and $n + 1$ weights, the formula will be exact for polynomials of degree at most $2n$. This does not need a proper proof, since this quadrature is constructed from Gaussian quadrature, which was proved in class.