Universidad de Costa Rica
Escuela de Matemáticas
MA-501 Análisis Numérico
Tarea 2

Juan Ignacio Padilla B.
Carné: B55272
Prof. Juan Gabriel Calvo
28 de enero de 2026

Tarea 2

Problema 1.

(Evaluando polinomios) Considere un polinomio de grado n escrito en la base usual

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

y defina el vector de coeficientes

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

Para un valor x_0 dado, deseamos evaluar $p(x_0)$. Para ello, consideramos 3 métodos diferentes:

Alg. 1) Mediante evaluación directa, calculamos cada término $a_k x^k$ y luego sumamos todos los términos.

Alg. 2) Defina la sucesión $\{b_k\}_{k=0}^{n}$ mediante la recursión hacia atraás

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

Entonces $p(x_0) = b_0$

Alg. 3) Utilizar el comando `polyval()` de `MATLAB` para evaluar el polinomio.

(a) Determine el número de operaciones (sumas y multiplicaciones) que se deben realizar en el Algoritmo 1. Tenemos:

$$\begin{aligned} a_0 &\rightarrow 0 \text{ operaciones.} \\ a_1 x &\rightarrow 1 \text{ operación.} \\ a_2 x^2 &\rightarrow 2 \text{ operaciones.} \\ &\vdots \\ a_n x^n &\rightarrow n \text{ operaciones.} \end{aligned}$$

Entonces en total tenemos que efectuar $n(n+1)/2$ multiplicaciones. Al tomar en cuenta las n sumas que hay que efectuar, se obtiene un total de

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

operaciones.

(b) Determine el número de operaciones (sumas y multiplicaciones) que se deben realizar en el Algoritmo 2.

Observe que en cada iteración $b_k = a_k + b_{k+1}x_0$ se necesitan 2 operaciones (salvo en la primera). Multiplicando por el número de iteraciones (n, pues de 0 a $n-1$ hay n números), se tiene que se necesitan $2n$ operaciones para calcular b_0.

(c) Para el Algoritmo 2, demuestre que $p(x_0) = b_0$.

Vamos a proceder por inducción sobre el grado de p. Para $n = 0$, la situación es trivial dado que si $p(x) = a_0$, la primera iteración $b_0 = a_0$ cumple que $p(x_0) = b_0$. Suponga la validez del resultado para polinomios de grado $n-1$ y sea

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

Considere el polinomio

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

Aplicando la fórmula recursiva para $q(x)$

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

Obtenemos por hipótesis de inducción que $c_1 = q(x_0)$. Ahora, tomando la recursión para $p(x)$,

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

Vemos que $b_k = c_k$ para $k = 1, \ldots, n$, pues las recursiones coinciden. Falta ver qué sucede cuando $k = 0$. Se tiene que

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

(d) Compare el tiempo de ejecución de los Algoritmos 1, 2 y 3 en `MATLAB`. Para ello, genere un vector con entradas aleatorias de una distribución uniforme en $(0, 1)$ para los coeficientes mediante el comando

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

Utilice $n = 10^8$, $x_0 = 0{,}1$ y compare los tiempos de ejecución.

Usando `MATLAB`, al aplicar el algoritmo 1 a un vector aleatorio de tamaño 10^8, el cual genera un polinomio de grado $10^8 - 1$, **se obtuvo un tiempo promedio de $11,6280274$ segundos** (se efectuó la prueba 5 veces).

Para el caso del algoritmo 2, las operaciones tardaron **un promedio de $0,2776524$ segundos** (en 5 pruebas). Esto representa una enorme mejoría con respecto al algoritmo 1, lo cual se puede deber a que el número de operaciones efectuadas es lineal con respecto a n, mientras que en el algoritmo 1, es de orden cuadrático.

Finalmente, para el algoritmo 3, **tomó en promedio 1, 2224204 segundos**(en 5 pruebas igualmente). Pareciera que, en esta situación, el Algoritmo 2 es el más rápido. Sin embargo, no se puede asumir esto en general, puesto que nuestros coeficientes son todos positivos y menores a 1. Tal vez con otra naturaleza de coeficientes, se tengan resultados distintos.

Nota: Al tener polinomios aleatorios, no se especifica el valor de $p(x_0)$, sin embargo, en el script adjunto se puede verificar que en efecto los valores coinciden para los 3 algoritmos.

(e) Considere ahora `a= ones(n,1)`, $n = 10^8$, $x_0 = 0,1$. ¿Cuál es el valor exacto de $p(x_0)$? Utilice el algoritmo 1 y 2 para obtener aproximaciones $\tilde{p}_1$ y $\tilde{p}_2$, respectivamente. ¿Cuántas cifras significativas son correctas para $\tilde{p}_1$ y $\tilde{p}_2$?

Calculemos el valor exacto de $p(x_0)$.Tenemos

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

Entonces

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

El cual se puede calcular usando la fórmula geométrica para obtener

$$p(0,1) = \left(\frac{1-\left(\frac{1}{10}\right)^{10^8}}{1-\frac{1}{10}}\right) = \frac{10^{10^8}-1}{9\times 10^{10^8-1}}$$

O más fácilmente podemos ver que

$$\begin{aligned} p(0,1) &= 1 + 0,1 + 0,001 + \cdots + \underbrace{0,0\cdots 01}_{10^{10^8}\text{ ceros}} \\ &= \underbrace{1,1\cdots 1}_{10^{10^8}\text{ unos}} \end{aligned}$$

El resultado del Algoritmo 1 es $\tilde{p}_1 = 1{,}111111111111111$, por lo cual es exacto con 16 cifras significativas. Mientras que el resultado de aplicar el Algoritmo 2 es también $\tilde{p}_2 = 1{,}111111111111111$, el cual es exacto también con 16 cifras significativas. Se concluye por lo tanto que ambos algoritmos tienen una alta exactitud. Por razones obvias, no es lógico esperar que `MATLAB` logre arrojar los 100 millones de decimales que posee el valor exacto de $p(x_0)$, sin embargo las aproximaciones dadas por los algoritmos se acercan de manera decente.

Problema 2.

Suponga ahora que realizamos un cambio de base y expresamos el polinomio en la base de los polinomios de Chebyshev,

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

Donde $T_k(x) = \cos(k \arccos x)$. En clase demostramos la relación de recurrencia

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

Deseamos calcular $p(x_0)$. Para ello, considere el siguiente algoritmo:

Alg 4) Defina la sucesión $\{b_k(x_0)\}_{k=0}^{n+2}$ mediante la recurrencia hacia atrás

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

Se tiene entonces que el valor deseado es

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

a) Determine el número de operaciones que se deben realizar en el Algoritmo 4.

Tenemos en cada iteración 4 operaciones. Al tener exactamente n iteraciones, se tendrá un total de $4n$ operaciones.

b) Verifique que la fórmula ($\star$) es correcta.

Note primero que $T_0(x) = 1$, y $T_1(x) = x$. De manera similar, procedemos por inducción fuerte sobre n. Si $p(x) = c_0 T_0 = c_0$, entonces la recursión simplemente toma $b_2 = b_1 = 0$ y

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

Ahora asumamos el resultado para combinaciones lineales de los primeros k polinomios de Chebyshev, con $k < n$. Sea

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

y definimos su recursión:

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

Considere ahora los polinomios

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

Para q, definimos la recurrencia

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

También definimos la recurrencia para r:

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

Tenemos, por hipótesis de inducción, que

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

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

Observe además que los primeros pasos de las recursiones para p, q y r coinciden. Más específicamente, $b_{k+1} = d_k$ para $k = 1, \ldots, n+1$ y $b_{k+2} = u_k$ para $k = 1, \ldots, n$. Esto nos permite observar que, volviendo a la recursión para p:

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

Y también que

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

Tenemos finalmente que

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

c) Escriba una función `y0=evalCheb(c,x0)` en `MATLAB` que calcule $y_0 = p(x_0)$ mediante la fórmula $(\star)$, donde la entrada es el vector de coeficientes $\mathbf{c}$ y el valor x_0.

Se implementó la función `y0=evalCheb(c,x0)`, adjuntada en los archivos.

d) Utilice `c=rand(n,1)`, con $n = 10^8$, $x_0 = 0,1$ y compare el tiempo de ejecución con la pregunta (1d). Comente

Al aplicar la función `y0=evalCheb(c,x0)`, a 5 vectores aleatorios de coeficientes entre 0 y 1, de tamaño 100 millones, se obtuvo un **tiempo de evaluación promedio de $\mathbf{0,3781628}$ segundos**, el cual es sustancialmente más rápido que los algoritmos 1 y 3. Sin embargo, es ligeramente más lento que el algoritmo 2. Todo esto se puede volver a justificar al echar un vistazo al número de operaciones que necesita cada uno de los algoritmos. Si recordamos que el 1 era de orden cuadrático con respecto a n, no es una sorpresa que resulte el más lento, pues el 3 y el 4 son de orden lineal, siendo el 4 un poco más lento, pues necesita el doble de operaciones que el 2. Una vez más, al no saber exactamente qué operaciones usa `polyval()`, no es posible una discusión de este tipo, solo se puede decir que tiene un tiempo de evaluación intermedio con respecto a los demás.

e) Considere el polinomio $p(x) = x^3 - 3x^2 + 1$. Exprese $p(x)$ como combinación lineal de polinomios de Chebyshev

$$p(x) = c_0T_0(x) + c_1T_1(x) + c_2T_2(x) + c_3T_3(x)$$

y verifique el comportamiento del Algoritmo 4 para $x_0 = 1$.

Tenemos que $T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1$ y $T_3(x) = 4x^3 - 3x$. Entonces basta con resolver el sistema de ecuaciones en 4 variables $\alpha, \beta, \gamma, \delta$

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

En donde fácilmente se ve que $\alpha = 1/4, \beta = -3/2, \gamma = 3/4$ y $\delta = -1/2$. Al aplicar el Algoritmo 4 en la base de Chebyshev, se obtiene el resultado $p(1) = -1$, el cual verifica el funcionamiento del algoritmo numéricamente.

Problema 3

(Integrando expansiones de Chebyshev) Con la notación del ejercicio anterior, considere el polinomio

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

a) Demuestre que para todo entero $n \in \mathbb{N}$,$n \neq 1$,

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

Deduzca que

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

Solución: Si $n = 0$, el resultado es trivial, puesto que $T_0 = 1$. Para $n > 1$, considere

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

Diferenciamos para obtener

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

De la misma forma para $T_{n-1}(x)$, se tiene

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

Sea ahora $\theta = \arccos(x)$. Al sustraer las expresiones anteriores, y aplicando algunas identidades trigonométricas, obtenemos

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

Integramos esta identidad

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

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

Ahora, note que para todo n,

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

Por lo tanto

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

Que es lo que se buscaba.[1] Ahora, para $p_n(x) = \sum_{k=0}^{n} c_k T_k(x)$ Simplemente se tiene que

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

b) Programe una función `I = intCheb(c)`, cuya entrada es el vector de coeficientes

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

dados en (3), que calcule el valor I de la integral (4).

Se implementó la función `intCheb(c)`, la cual se adjunta en los archivos.

Problema 4.

Dado $n \in \mathbb{N}$, considere los $n+1$ puntos de Chebyshev dados por

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

Dada una función f es posible escribir el polinomio de interpolación de Lagrange con nodos $\{x_j\}_{j=0}^{n}$ en la base de polinomios de Chebyshev

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

Al conocer los valores $\{f(x_k)\}_{j=0}^{n}$ podemos calcular los coeficientes c_k de manera eficiente mediante la inversa de la transformada discreta de Fourier, la cual se implementa con el comando `ifft` en `MATLAB`. La siguiente función recibe como entrada los valores $\{f(x_k)\}_{j=0}^{n}$ en un vector columna y retorna los coeficientes en un vector $\mathbf{c}$.

[1] Los cálculos que se omitieron son sencillos.

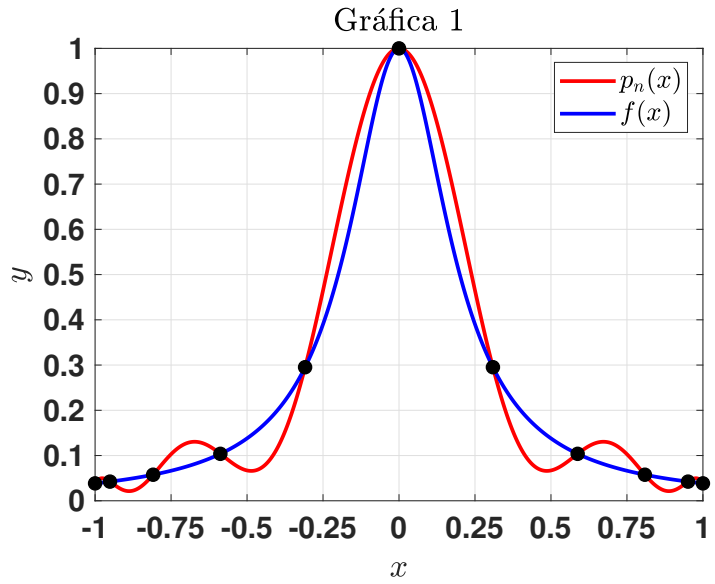
```
function c= val2coef(values)
n=size(values,1);
% Mirror the values
tmp=[values(n:-1:2,:);values(1:n-1,:)];
% apply ifft
c=real(ifft(tmp));
%Truncate
c=c(1:n,:);
%Scale the interior coefficients
c(2:n-1,:)=2*c(2:n-1,:);
end
```

a) Implemente una función `[coef,nodos] = interpCheb(n,f)` que reciba el grado n y la función f, que calcule los nodos de Chebyshev y los coeficientes $\{c_k\}$.

Se implementó la función `[coef,nodos] = interpCheb(n,f)`, adjuntada en los archivos.

b) Para $f(x) = 1/(1+25x^2)$, $n = 10$, calcule los coeficientes `c=interpCheb(n,f)`. Defina `xx = linspace(-1,1,1e3)` y utilice el comando `evalCheb` para evaluar p_n en cada entrada de `xx`. Grafique f y p_n en un mismo gráfico, incluyendo los nodos de interpolación. Comente su resultado. ¿Ocurre el fenómeno de Runge?

Al seguir las instrucciones, se obtiene la figura 1.

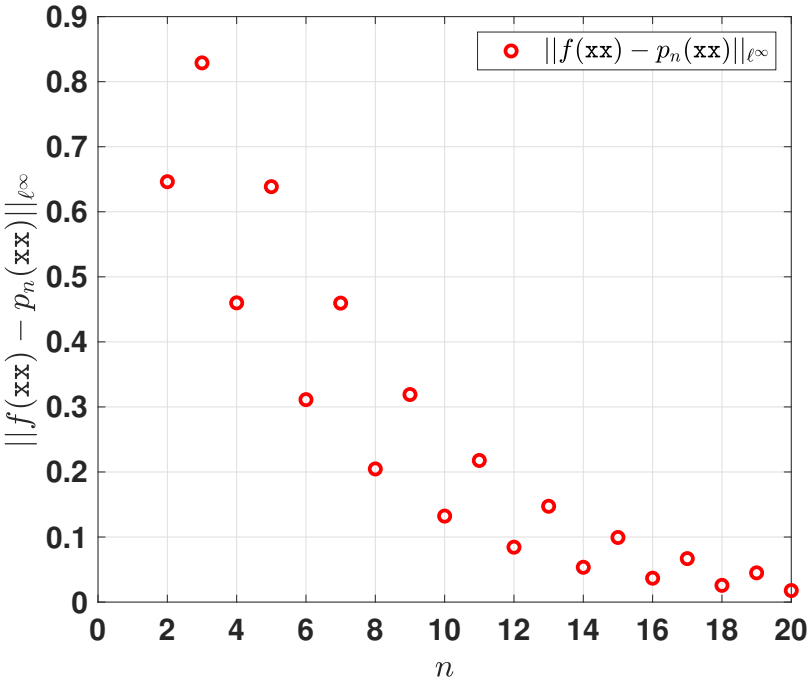


Para $n = 10$, la aproximación no es demasiado buena, pues se tiene cierto grado de fluctuación entre los valores de $p_n(x)$ y $f(x)$. Podemos ver, sin embargo, que en los nodos de Chebyshev (los puntos negros), las curvas de hecho coinciden. Se concluye que para $n = 10$, pareciera que el fenómeno de Runge **no ha desaparecido del todo.**

c) Grafique $||f(\texttt{xx}) - p_n(\texttt{xx})||_{\ell^\infty}$ en función de n para $n \in \{2, \ldots, 20\}$.¿Qué velocidad de convergencia observa conforme n aumenta?

Al graficar el error, se obtiene la Gráfica 2:

Gráfica 2



Se observa que el error alterna, siendo ligeramente mayor para n impar, esto podría deberse a la simetría de la función (al escoger nodos asimétricos, los valores de interpolación serán a su vez asimétricos). Sin embargo, independientemente de la fluctuación, el error disminuye conforme n aumenta, con una velocidad que pareciera ser **superlineal**.

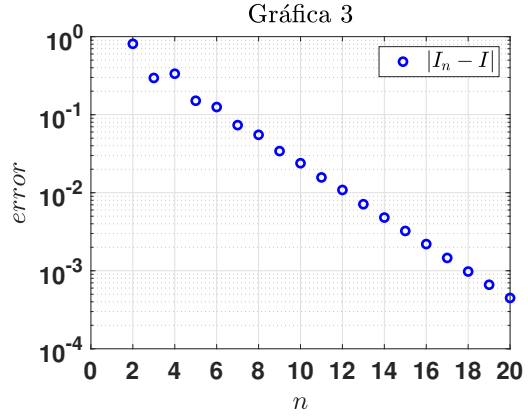
d) Para $f(x) = 1/(1+25x^2), n = 10$ utilice la fórmula 3a) para aproximar $\int_{-1}^{1} f(x)dx$. Grafique el error absoluto en función de n para $n \in \{2, \ldots, 20\}$. ¿Qué velocidad de convergencia observa conforme n aumenta?

Al aplicar la función `intCheb` a f con $n = 10$, se obtiene que

$$\int_{-1}^{1} f(x)dx \approx 0{,}573232153266308$$

El cual se aproxima al valor exacto de $\frac{2}{5}\arctan 5 \approx 0{,}549360306778006344$ con 1 decimal. Para diferentes n tenemos la gráfica 3.

Gráfica 3



En donde I_n es la aproximación de la integral I, usando polinomios en base de Chebyshev, de dimensión n.

Se puede observar que la velocidad de convergencia es casi perfectamente **lineal**, por lo cual la integración por coeficientes de Chebyshev parece una buena opción numérica.

e) ¿Qué ventajas observa en interpolar en nodos de Chebyshev con expansiones en polinomios de Chebyshev para aproximar integrales?.

La principal ventaja de interpolar usando los nodos de Chebyshev es el hecho de que el **fenómeno de Runge desaparece** para valores altos de n, esto elimina las fluctuaciones que puedan aparecer en los extremos del intervalo de interpolación, lo cual a su vez mejora la confiabilidad de la integral. Como una ventaja secundaria, se tiene la facilidad con la que se computan estas integrales. Al expresar el polinomio en la base de Chebyshev (lo cual no representa grandes dificultades computacionales, pues solo se trata de un cambio de base), se tiene que la integral del polinomio de interpolación viene dada por la fórmula

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

La cual es sencilla de calcular, pues solo involucra las coordenadas del polinomio en la base, y sumas, multiplicaciones, y divisiones. Se concluye por lo tanto que los polinomios de Chebyshev representan una útil herramienta en análisis numérico, tanto en interpolación como en integración de funciones suaves.

Problema 5.

Construya la fórmula de cuadratura

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

donde hemos fijado el nodo $x_0 = a$; esto es, calcule los pesos $\{W_k\}_{k=0}^n$ y nodos $\{x_k\}$ de manera que la fórmula anterior sea exacta para polinomios de grado a lo sumo $2n$. Sugerencia: Escriba

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

con $r \in \mathbb{R}$.
Siguiendo la sugerencia, vamos a asumir que f es un polinomio de grado $2n$ (se podría interpretar como algún polinomio que la interpole). Note que, evaluando en a, tenemos inmediatamente que $r = p_{2n}(a)$.
Considere el peso

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

Podemos aplicar la cuadratura de Gauss a $q_{2n-1}(x)$ para tener

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

Donde $\{\tilde{x}_k\}_{k=1}^n$ son los nodos de la cuadratura de Gauss aplicada al peso $\tilde{w}$, y $\{\tilde{W}_k\}_{k=1}^n$ sus respectivos pesos.
Como

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

Multiplicando por $w(x)$, e integrando se tiene

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

Pero como $q_{2n-1}(\tilde{x}_k) = \frac{p_{2n}(\tilde{x}_k)-p_{2n}(a)}{\tilde{x}_k-a}$, tenemos

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

Por lo que podemos tomar

$$\begin{aligned}W_0 &= \int_a^b w(x)dx - \sum_{k=1}^{n} \frac{\tilde{W}_k}{\tilde{x}_k - a} \\ W_k &= \frac{\tilde{W}_k}{\tilde{x}_k - a} \quad \text{para} \quad 1 \leq k \leq n\end{aligned}$$

Y tomando $x_k = \tilde{x}_k, k = 1, \ldots, n$ se obtiene la cuadratura deseada. Observe que $W_k > 0$, pues $\tilde{W}_k > 0$. Para $k = 0$, defina $P(x) = (b - x)\prod_{k=1}^{n}(x - x_k)^2$, la cuadratura que construímos inmediatamente arroja

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

Por lo que $W_0 > 0$. Finalmente, al haber definido n nodos y $n + 1$ pesos, la fórmula será exacta para polinomios de grado a lo sumo $2n$. Esto no necesita demostración propiamente, puesto que esta cuadratura se construye a partir de la cuadratura de Gauss, la cual se demostró en clase.