Universitat autònoma de Barcelona

Grau en matemàtiques

Memòria de la pràctica 1

Mètodes numèrics

Aleix
Suarez Sayabera

Abril 2024

Índex

1 Introducció

L'objectiu d'aquesta pràctica és experimentar amb diferents algorismes per determinar l'estabilitat numèrica i eficiència d'aquests. En la pràctica estudiaré quatre algorismes diferents que calculen el sinus i cosinus dels múltiples d'un angle.
Per tal d'aconseguir això, he hagut de programar aquests 4 algoritmes, i fer programes per calcular els errors comesos i el temps de càlcul.

2 Implementació del software

Aquests quatre algorismes els hem hagut de programar a C. Cada algorisme rebia un enter k, un double x, i dos vectors c i s, que corresponen als valors del cosinus i sinus, on a cada component estava el múltiple de l'angle x per j, la component del vector. L'enter k representa fins a quin múltiple de l'angle x volem calcular, x és el double que li fiquem a la funció.

La funció en si, per als quatre algorismes no retorna res, són totes del tipus void, el que fa és omplir els vectors c i s amb els valors del sinus i cosinus corresponents, es a dir, el vector c a la component j, tindrà el valor del cosinus de $j \cdot x$.

2.1 Programació dels algorismes

2.1.1 Trigrec0

El primer algorisme que es demana programar és trigrec0. Aquest algorisme com he explicat, rep un enter k, un double x, i dos vectors c i s, que els passaré com a apuntadors per un millor ús de la memòria.
Aquest primer algorisme calcula els sinus i cosinus directament amb la funció sin() i cos() de C. Per tant, simplement amb un bucle for es pot programar aquest algorisme.

2.1.2 Trigrec1

El següent algorisme calcula els valors d'una forma diferent, ho fa de forma recurrent utilitzant que:

$$\cos((m+1)x) = 2\cos(x)\cos(mx) - \cos((m-1)x)$$
$$\sin((m+1)x) = 2\cos(x)\sin(mx) - \sin((m-1)x)$$

D'aquesta forma els c_{j+1}, s_{j+1} es calculen a partir dels c_j, s_j. I pot arribar a ser un algorisme més ràpid, però on en comptes de fer solament dues operacions elementals per component dels vectors, es fan moltes més, i es pot arribar a tenir inestabilitat numèrica, cosa a veure més endavant a la pràctica.
Per programar això es fa de forma similar a trigrec0, amb un bucle for, però amb la diferència de que hem de donar el valor de les components 0 i 1 als vectors c i s, on $c_1 = cos(x)$ i $s_1 = sin(x)$, i tots els altres c_j dependran de les dues primeres components, on clarament $c_0 = 1$ i $s_0 = 0$.

2.1.3 Trigrec2

El tercer algorisme és molt similar a l'anterior, però intenta solucionar l'error de la inestabilitat numèrica de Trigrec1, això ho fa utilitzant les fòrmules de sinus i cosinus de la suma sobre $\cos((m+1)x)$, $\sin((m+1)x)$, aixi obtenint:

$$\cos((m+1)x) = \cos(mx)\cos(x) - \sin(mx)\sin(x),$$
$$\sin((m+1)x) = \sin(mx)\cos(x) + \cos(mx)\sin(x).$$

Per programar això ho he fet quasi igual a Trigrec1, amb un bucle for i declarant les dues primeres components de cada vector. La diferència amb Trigrec1 és la recurrència que fa servir.

2.1.4 Trigrec3

L'últim algorisme és Trigrec3, aquest algorisme utilitza un altre cop una forma recurrent per calcular els valors de s_j i c_j. El que s'intenta millorar amb aquest algorisme és l'estabilitat numèrica, per tal de fer això es considera la diferència entre sinus consecutius δ_j^s i cosinus consecutius δ_j^c, d'aquesta manera:$\delta_{m+1}^c := \cos((m+1)x) - \cos(mx)$ $\delta_{m+1}^s := \sin((m+1)x) - \sin(mx)$. Amb això i juntament amb un paràmetre $t := 2\delta_1^c$, tenim aquesta recurrència:

$$\begin{aligned}
\cos((m+1)x) &= \cos(mx) + \delta_{m+1}^c \\
\sin((m+1)x) &= \sin(mx) + \delta_{m+1}^s \\
\delta_{m+2}^c &= t\cos((m+1)x) + \delta_{m+1}^c \\
\delta_{m+2}^s &= t\sin((m+1)x) + \delta_{m+1}^s
\end{aligned}$$

A l'hora de programar això he declarat de forma especial les variables delta per aconseguir un algorisme més ràpid. Primerament, no he declarat les deltes com a vectors, sinó com a doubles, on aquests van canviant per cada pas del bucle for. A l'hora de delcarar-los, en comptes de fer-ho directament per com estaven definits, $\delta_1^c = -2(\sin(x/2))^2$, he declarat primer delta com $\sin(x/2)$ i després he declarat $\delta^c = -2 * \delta^c * \delta^c$. Per fer la delta del sinus, en comptes d'utilitzar la funció POW, per fer la potència de -1, he comprovat la paritat de la part entera de x/π, ja que això és més fàcil de fer per a l'ordinador.

2.2 Programa que imprimeix els valors

Per tal de fer aquest programa, trigrec_escr, que el que fa és imprimir els valors per pantalla dels 4 algorismes anteriors, com que és un altre fitxer c, i cal implementar les funcions trigrec, en comptes de tornar-les a escriure a trigrec_escr es pot fer servir un fitxer.h, que contengui les declaracions de les funcions i incloure-ho a trigrec_escr.

Per fer aquest programa cal declarar dos vectors per a cada algorisme, un del sinus i un altre del cosinus. Per això, com tot ho he fet amb apuntadors, declarant cada c_i i s_i amb un double*, i amb el malloc, ja estaria fet, tot amb mida $k+1$. Ara per imprimir els resultats dels algorismes, abans cal aplicar les funcions trigrec als vectors corresponents, que són 2 per cada algorisme, per tant 8, i un cop ja estiguin els vectors amb els seus valors corresponents, amb un bucle for he fet que s'imprimeixin els resultats.

2.3 Càlcul dels errors i del temps d'execució

El següent programa que es demanava fer és trigrec_temps_err, el que fa aquest programa és calcular el temps que triga cada algorisme a calcular els sinus i cosinus, i també quin és l'error màxim que hi ha de càlcul per cada algorisme, solament dels tres últims, ja que el primer algorisme fa els càlculs exactes, exactes tenint en compte les limitacions dels ordinadors.
Per fer aquest programa, com abans, he hagut de declarar els vuit vectors en forma d'apuntador amb el malloc. Ara, per calcular el temps d'execució de cada algorisme, he utilitzat la funció temps que se'ns ha proporcionat, per utilitzar-la només calia declarar uns doubles, un amb el temps inicial, que serà la referència de quant es triga, i dos per a cada algorisme, un és declara la línia abans de la crida i l'altre després, juntament es crida la funció temps a cada variable, ja que així la variable indica el temps que passa entre t_0 i l'altre t_i. Ara, la diferència entre els dos valors és el temps d'execució. Això es fa per cada un dels quatre algorismes, i imprimint per pantalla aquestes diferències ja està feta aquesta part del programa. Per declarar la funció temps, com amb els algorismes trigrec, s'utilitza un fitxer h.

Per calcular els errors màxims que es fan en cada vector dels últims tres algorismes, he utilitzat un bucle for. Com a referència d'un càlcul perfecte he usat els valors de c_j^0 i s_j^0 que són els vectors calculats per l'algorisme trigrec0, que són estables numèricament, ja que estan calculats amb les funcions cosinus i sinus. En el bucle for, per calcular els errors màxims de cada algorisme, he calculat per a cada j entre 0 i k, la diferència entre c_j^0 i c_j^i per $i = 1, 2, 3$, i d'igual forma amb els sinus, cada i representa els diferents algorismes. En el bucle for, declaro dos tipus de variables per a cada vector, una per l'error màxim i l'altra per l'error que es calcula en cada pas. La variable que es fa córrer en el bucle és j, que és la que utilitzo per a les

components dels vectors. Ara, l'error que es calcula en cada pas és aquesta resta entre el valor del cosinus o sinus exacte i el de l'algorisme en la component j, amb aquest error, aplico valor absolut, i si aquest error és més gran que l'error màxim, que l'he inicialitzat amb zero, aleshores l'error calculat és el nou error màxim, i així successivament fins a les $k+1$ components.
Al final, imprimeixo aquests errors màxims de cada algorisme.

3 Experiments numèrics

Ara cal experimentar amb l'últim programa, que és el que calcula els errors i el temps d'execució. Passaré al programa diferents valors i experimentaré amb el temps d'execució i els errors numèrics comesos.

3.1 Estudi dels errors

Per tal d'experimentar amb els algorismes, veuré els resultats del programa per a aquests valors de k i x:

k	x
100	0.0123
1000	0.00123
10000	0.000123
100000	0.0000123
1000000	0.00000123
10000000	0.000000123

Taula 1: Valors de k i x.

Que per $k = 100$ i $x = 0.0123$ han donat com a resultat:

algorisme	temps d'execució	error cosinus	error sinus
Trigrec0	instantani	0	0
Trigrec1	instantani	2.2526×10^{-13}	1.2712×10^{-13}
Trigrec2	instantani	1.3322×10^{-15}	2.8865×10^{-15}
Trigrec3	instantani	3.3306×10^{-16}	2.2204×10^{-16}

Taula 2: Valors per $k = 100$ i $x = 0.0123$.

Al ser una k relativament petita el temps d'execució és mínim, i el programa el retorna com zero. Analitzant els errors, encara que siguin molt petits, el pitjor algorisme és el segon, i el millor, sense tindre en compte, el primer, és el quart. Ara prenent valors més grans:

algorisme	temps d'execució	error cosinus	error sinus
Trigrec0	instantani	0	0
Trigrec1	instantani	3.5166×10^{-7}	1.6147×10^{-7}
Trigrec2	instantani	2.0863×10^{-12}	4.4283×10^{-12}
Trigrec3	instantani	7.6605×10^{-15}	9.3258×10^{-15}

Taula 3: Valors per $k = 100000$ i $x = 0.0000123$.

Ara es veu molt clarament la diferència dels algorismes en termes de la inestabilitat numèrica, l'error de l'últim algorisme és mínim, mentre que el segon algorisme és molt inestable numèricament. Encara per aquests valors no es pot apreciar quin dels quatre és més ràpid.
Pels valors més grans de la taula, el programa dona com a resultat:

algorisme	temps d'execució	error cosinus	error sinus
Trigrec0	0.046 segons	0	0
Trigrec1	0.063 segons	0.001	0.0005
Trigrec2	0.047 segons	6.2405×10^{-11}	1.3219×10^{-10}
Trigrec3	0.0469 segons	1.1879×10^{-13}	7.5828×10^{-14}

Taula 4: Valors per $k = 10000000$ i $x = 0.000000123$.

Per a aquests valors el programa ja retorna temps d'execució dels algorismes, on els tres per a aquests valors són bastant similars. En termes de l'estabilitat numèrica, l'algorisme trigrec1 és molt dolent, mentre que trigrec2 i trigrec3, es mantenen bastant bé.
Ara bé, els algorismes funcionen diferent segons el tipus de valors que passem al programa, els valors de la taula compleixen que $k \times x = 1.23$, i per càlculs de com es propaguen els errors en cada algorisme, per aquests valors, l'algorisme trigrec1 és molt inestable, mentre que per a aquests tipus de valors, l'últim algorisme, trigrec3, és molt estable numèricament.
Ara passaré valors amb k molt gran i x també gran, per veure el comportament dels algorismes:

algorisme	temps d'execució	error cosinus	error sinus
Trigrec0	4.796 segons	0	0
Trigrec1	0.516 segons	1.2523×10^{-7}	1.2521×10^{-7}
Trigrec2	0.531 segons	1.2054×10^{-7}	1.2054×10^{-7}
Trigrec3	0.344 segons	1.2714×10^{-7}	1.2714×10^{-7}

Taula 5: Valors per $k = 100000000$ i $x = 13.4758$.

Per a aquests valors ja es pot veure clarament la diferència de velocitat, el primer algorisme per als valors de la taula era pràcticament igual de ràpid que els altres tres, mentre que ara es veu que la diferència és molt gran, arribant a ser 9 vegades més lent que els altres. L'algorisme més ràpid per a aquests valors ha sigut trigrec3, i en termes de l'estabilitat numèrica tots tres són bastant similars, ara com $x \times k$ no és proper a un, l'últim algorisme ja no és tan bo.
També s'ha de tindre en compte que com els tres algorismes que utilitzen recurrències treballen amb una aproximació inicial, a l'hora de fer càlculs l'error s'anirà multiplicant. On aquest error variarà segons el numero de condició de cada algorisme, que també varia segons el valor de k i x que li passem al programa.

3.2 Velocitat de càlcul

Ara estudiaré quina és la velocitat de càlcul del meu ordinador. Primer veure el temps de càlcul dels dos últims algorismes, i tenint en compte el nombre d'operacions que han de fer segons k, estimaré la quantitat d'operacions per segon que fa el meu ordinador.

A l'algorisme trigrec2, per a cada pas del bucle for, es fan 6 operacions, quatre multiplicacions i dos sumes. On com les dues primeres components de cada vector, ja hi estan ocupades, es fan $(k+1-2) \times 6$ operacions en total per qualsevol k. A l'algorisme trigrec3, a cada pas del bucle for es fan també 6 operacions, però en aquest cas són quatre sumes i dues multiplicacions. En aquest algorisme es fan també operacions abans del bucle for, però a l'hora d'estimar la velocitat de càlcul són indiferents per a k gran. Per tant, a l'algorisme trigrec3 posaré que es faran $(k+1-2) \times 6$ operacions en total, encara que són una mica més.

Tenint en compte els resultats de la taula 4 i 5, i el nombre d'operacions fetes, obtenim aquests valors per al temps d'operacions per segon:

temps d'execució	valor de k	operacions fetes	operacions per segon
0.047 segons	10000000	59999994	1276595617
0.0469 segons	10000000	59999994	1279317569
0.531 segons	100000000	599999994	1129943492
0.344 segons	100000000	599999994	1744186029

Amb això obtenim una mitjana de 1357510677 operacions per segon que fan els algorismes trigrec2 i trigrec3 al meu ordinador.

Ara tenint en compte aquesta estimació de les operacions que fa el meu ordinador per segon, i el temps de càlcul de trigrec0, podem estimar a quantes operacions aritmètiques equival utilitzar la funció sin i cos de c.

Tenint en compte que per $k = 100000000$ trigrec0 ha trigat 4.796 segons, i que s'ha cridat a la funció sin i cos k vegades cadascuna per fer el càlcul, juntament amb les dues multiplicacions que es fan a cada pas de trigrec0. Tenim que, utilitzar la funció sin i cos equival a fer $(4.796 \times 1357510677 - 2k)/2k$ operacions, i per tant una mitjana de 32 operacions aritmètiques per cada crida en l'algorisme trigrec0, això tenint en compte les dades de l'última tabla. Per tindre una millor estimació caldria utilitzar-ne moltes més dades.