MALMÖ HÖGSKOLA

Inbyggda system och signaler
Digital signalbehandling

Labuppgift 3
1504c

Namn 1 Leonard Holgersson

Namn 2 Hadi Deknache

Tommy Andersson
februari 2017

Labuppgift 3, FIR-filter

Skriv inte ut detta dokument utan ha det öppet på datorn under laborationen och besvara frågorna direkt i dokumentet. En del foton skall tas och klistras in. Försök använda lämplig bildkvalitet så bilden syns tydligt men filstorleken inte blir för stor. Efter laborationen laddas dokumentet upp som pdf på Its learning.

Laborationen genomförs som vanligt i par dvs. ni jobbar två och två. Vid inlämningen på Its learning anges vem som jobbat ihop.

Utrustning

- Labplatta med uppkoppling från labuppgift 1
- Funktionsgenerator Wavetek
- Oscilloskop R&S HMO1002 med två probar
- 1 st koaxialkabel, 1 m med BNC-kontakter i ena änden och banankontakter i andra.
- Medhavd smartphone

OBS! Om du gjorde den frivilliga extra uppgiften i lab 2 bör du nu ta bort LP-filtret (annars måste du ta med det i beräkningarna framöver)

FIR-filter, glidande medelvärde

Kod för FIR-filter

Dags att skriva kod för ett FIR-filter. Utgå från koden till Labuppgift 1 (1504a). Den nya koden skall in i avbrottsrutinen, avsnittet för filterkod.

Utsignalen från ett FIR-filter ges av

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k]$$

Skriv koden så att filtrets längd *M* lätt kan varieras t.ex. genom att använda **#define** M ….
Definiera en array för de fördröjda värdena av insignalen, xbuff[], och en annan för koefficienterna, b[]. Båda arrayerna skall ha längden *M*+1.
Använd datatypen *static float* i detta första filter. Initiera arrayerna med 0 respektive aktuella värden för koefficienterna.
Börja koden med att ge plats för det senaste invärdet genom att flytta alla element i xbuff[] ett steg mot slutet: xbuff[k+1]=xbuff[k].
Men lägg märke till att man måste börja bakifrån…
Lägg nu in det inkommande värdet: xbuff[0] =(float)invalue.

(jämför gärna med den förkastade koden för fördröjning i Labuppgift 2)
Nu är det dags att bilda summan och att därefter till slut konvertera till typ *uint32_t* och att lägga värdet i outvalue.

Test av kod med glidande medelvärdesfilter

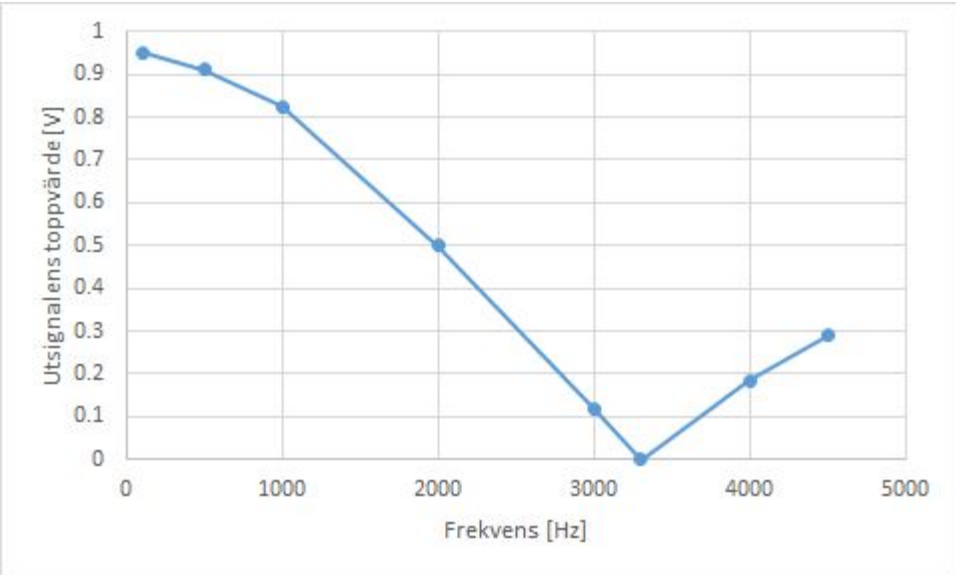
Börja med att testa med M=2 och koefficienterna b_0=0.3333, b_1=0.3333, b_2=0.3333 dvs. ett glidande medelvärdesfilter med tre termer.
Kör programmet och variera insignalens frekvens från 100 Hz till 5000 Hz (halva sampelfrekvensen). Var noga med att ställa in funktionsgeneratorn så att insignalens toppvärde blir 1 V. Mät speciellt utsignalens toppvärde för nedanstående värden.

Frekvens [Hz]	Utsignalens toppvärde [V]
100	0,95
500	0,91
1000	0,825
2000	0,5
3000	0,12
4000	0,184
4500	0,29
3300	0

När du långsamt ökar frekvensen från 100 till 5000 Hz kommer utsignalen att i stort sett bli noll vid en frekvens. Anteckna på sista raden vilken frekvens detta svarar mot.

Plotta ovanstående i ett exceldiagram med frekvensen som x-axel och utsignalen som y-axel. Glöm inte att sätta ut axelbeteckningar och enheter.

Klistra in nedan.



Frekvensfunktionen för ett FIR-filter med koefficienterna $b_1, b_2, \ldots b_M$ fås som

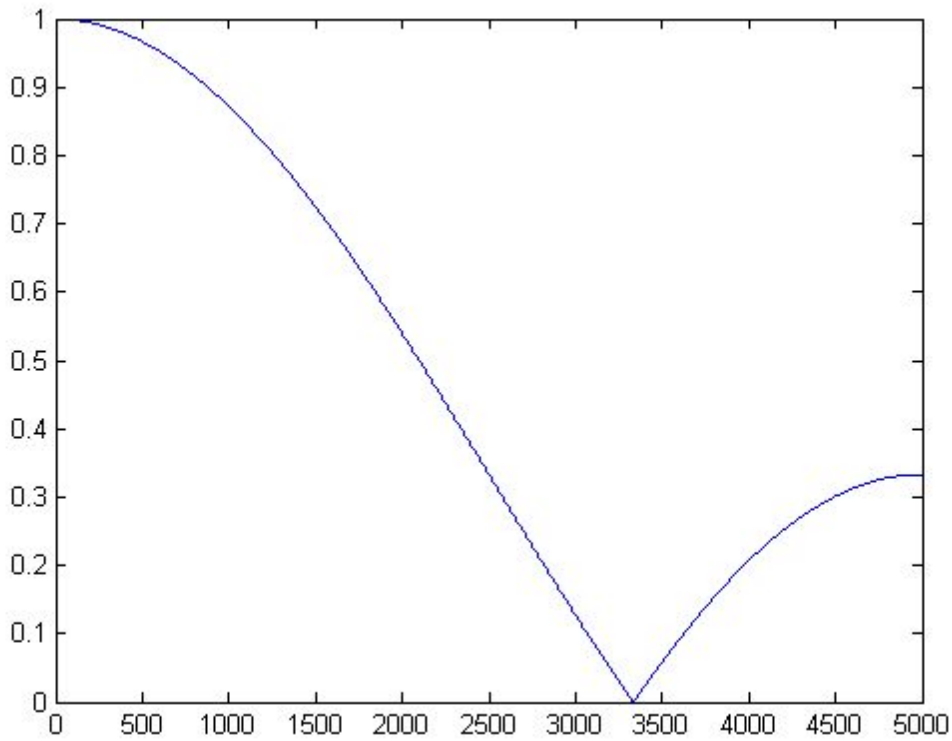
$$H(e^{j\hat{\omega}}) = \sum_{k=0}^{M} b_k \cdot \left(e^{j\hat{\omega}}\right)^{-k} = \sum_{k=0}^{M} b_k \cdot e^{-j\hat{\omega}k}$$

Använd Matlab för att plotta absolutbeloppet av frekvensfunktionen. Använd frekvensen *f* som variabel för att enkelt kunna jämföra med mätningarna. Plotta frekvensfunktionen i frekvensintervallet noll till halva sampelfrekvensen.

Så här kan Matlabkoden se ut:

```
fs=10000;
f=(0:1:5000);
omega=2*pi*f/fs;
H=0.3333+0.3333*exp(-i*omega)+0.3333*exp(-i*omega*2);
plot(f,abs(H))
```

Klistra in figuren här:



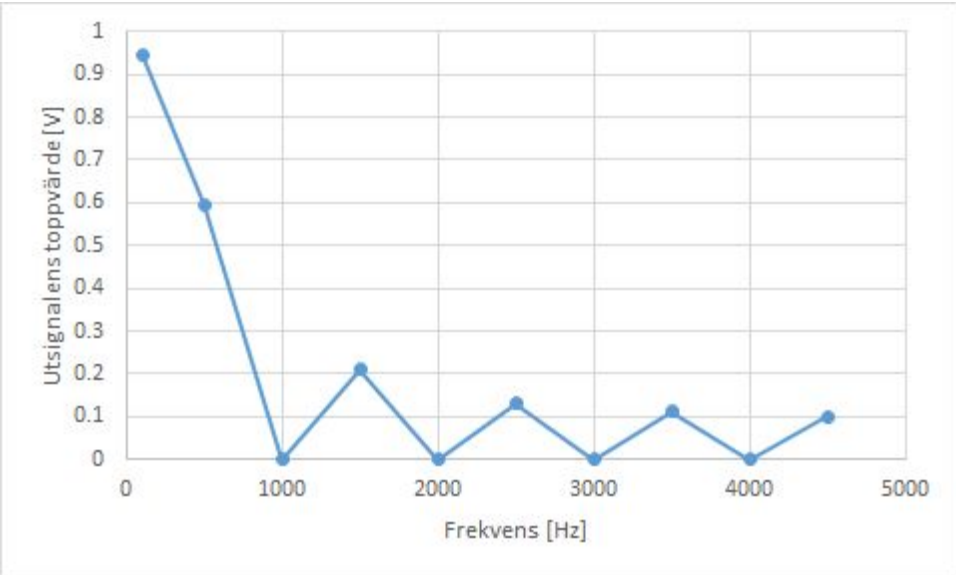
Jämför med de erhållna mätvärdena.
Kommentar:

Värden skiljer sig något litet, men ser ganska snarlikt ut som plotten i matlab när man jämför värden, vid de olika punkterna, med dem uppmätta värden.

Öka nu M till 9 och lägg in 10 värden i b[]. Koefficienterna lika även denna gång men nu måste de ges värdet 0.1 om ”förstärkningen” i filtret skall bli 1.
Variera frekvensen som tidigare och mät utsignalen.

Frekvens [Hz]	Utsignalens toppvärde [V]
100	0,945
500	0,595
1000	0
1500	0,211
2000	0
2500	0,132
3000	0
3500	0,112
4000	0
4500	0,1

Plotta ovanstående i ett exceldiagram som i förra uppgiften.
Klistra in:

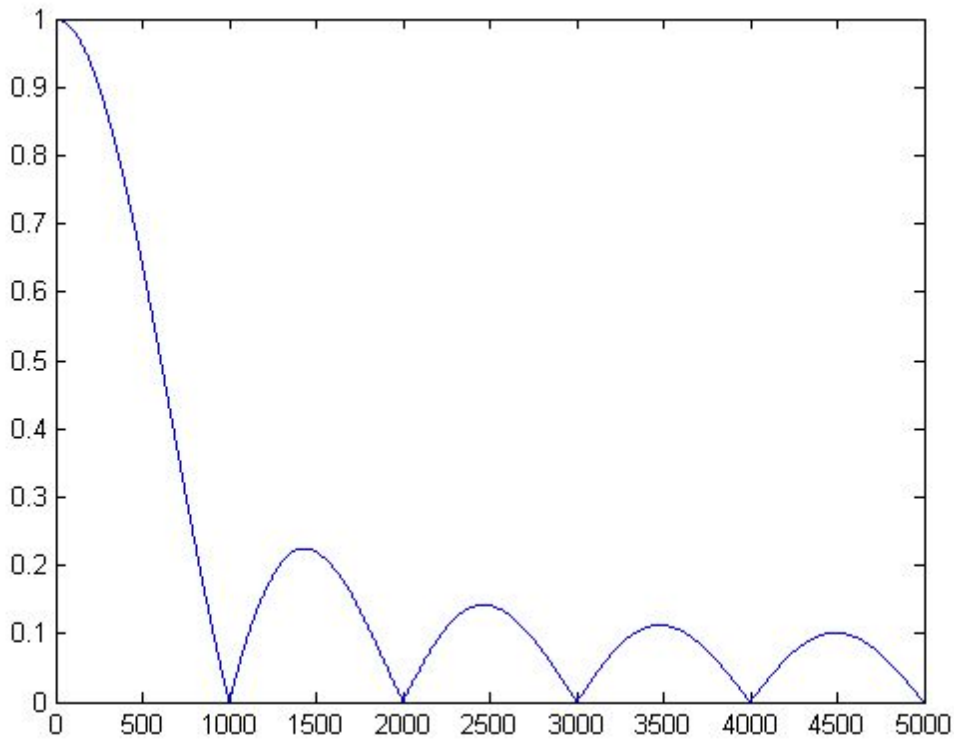


Försök även förklara varför utsignalen blir 0 för vissa frekvenser utan att använda frekvensfunktionen (ledning: det är ju medelvärdet av ett antal sampel som bildas):

Eftersom vi tar M+1 till 10 får vi under vår period på 10kHz 10 punkter vilket jämt fördelas över dessa dvs. 1kHz och därför ger 0V vid dessa vart 1kHz. Detta eftersom vid vissa samplingar tar dessa ut varandra och blir summan då 0. Två punkter sammanfaller vid olika positioner på kurvan där summan tar ut varandra.

Använd nu Matlab för att plotta absolutbeloppet av frekvensfunktionen. Modifiera Matlabkoden ovan för fallet M=9. Plotta frekvensfunktionen i frekvensintervallet noll till halva sampelfrekvensen.

Klistra in:



Kommentar:

Mätningarna stämmer väl överens med den teoretiska beräkningen!

Skulle det visa sig att mätningarna och den teoretiska beräkningen inte stämmer så är något fel! Kolla Matlabkoden, kolla processorkoden, backa och rätta till!

När du är övertygad om att allt är OK klistrar du in den del av C-koden du skrivit här (samma kod kommer att användas i denna och den återstående signalbehandlingslabben så den måste

vara rätt):

```
void TC0_Handler(void)
{
	static float xbuff[M+1] = {0};
	static float b[M+1]={0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};

	volatile uint32_t ul_dummy;
	uint32_t invalue, outvalue;
	float value=0;

	/* Clear status bit to acknowledge interrupt */
	ul_dummy = tc_get_status(TC0, 0);			//The compare bit is cleared by reading the register, manual p. 915

	/* Avoid compiler warning */
	UNUSED(ul_dummy);

	ioport_set_pin_level(CHECK_PIN,HIGH);		//put test pin HIGH

	adc_start(ADC);
	while((adc_get_status(ADC) & 0x1<<24)==0);  //Wait until DRDY get high

	invalue=adc_get_latest_value(ADC);			//get input value


	for (uint32_t k=M;k>0;k--)
	{
		xbuff[k]=xbuff[k-1];
	}

	xbuff[0]=(float)invalue;

	for (uint32_t x=0;x<M+1;x++)
	{
		value+=b[x] * xbuff[x];
	}

	outvalue=(uint32_t)value;


	dacc_write_conversion_data(DACC,outvalue);	//send output value to DAC

	ioport_set_pin_level(CHECK_PIN,LOW);		//put test pin LOW

}
```

Exekveringstid

Flyttalsberäkningar

Exekveringstiden för filteralgoritmer är viktig. Beräkningarna måste ju hinna utföras under ett sampelintervall (i vårt fall 0,1 ms eftersom sampelfrekvensen är 10 kHz)

Således är det dags att undersöka exekveringstiden.
Koppla oscilloskopets ena kanal till testutgången Digital Pin 22 och justera trigg och tidbas så det syns ett pulståg. Programmet lägger ju ut en etta när avbrottsrutinen startar och en nolla när avbrottsrutinen lämnas. Pulsen är alltså hög när processorn jobbar med filteravbrottet.

Kontrollera att avståndet mellan pulserna stämmer.

Mät upp tiden då en puls är hög (”zooma in” med tidbasen) och notera den.

Exekveringstid för M=9 (10 termer i summan): 27,91µs

Beräkna den ungefärliga beräkningstiden för en term (tänk på att A/D – D/A också tar tid, se labuppgift 1 (1504a)

Beräkning

I laboration 1 fick vi att beräkningstiden för A/D-D/A var 3,28 µs. Tar vi bort dessa 3,28 µs från de 27,91µs vi fick som exekveringstid nu borde det stämma överens med beräkningstiden för alla våra termer:

27,91-3,28 = 24,63 µs.

Vi har 10 termer, så den ungefärliga beräkningstiden för en term är $\frac{24,63}{10}$ *=2,463 µs*

Exekveringstid för en term 2,463 µs

Heltalsberäkningar

Det är enkelt att använda datatypen *float* men man kan misstänka att beräkningarna går fortare om man använder heltal.
Gör en ny variant av koden där datatypen *float* byts mot *int32_t*.
För att kunna hantera decimaltalen för koefficienterna måste dessa skalas om. Detta kan göras genom att multiplicera med t.ex. 10000 så i fallet M=9 blir b_0=1000, b_1=1000, b_2=1000 osv. .
Men det betyder förstås att man måste dividera med 10000 innan värdet läggs i outvalue.
Kör programmet och testa att det tycks fungerar som tidigare genom att variera insignalens frekvens och studera utsignalen. Gör detta för fallet M=9.
Du behöver inte göra ny tabell och graf men övertyga dig om att resultatet blir det samma som vid flyttalsberäkningarna.

Klistra in koden här (för fallet M=9):

```
void TC0_Handler(void)
{
	static uint32_t xbuff[M+1] = {0};
	static uint32_t b[M+1]={1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
1000};

	volatile uint32_t ul_dummy;
	uint32_t invalue, outvalue;
	uint32_t value=0;

	/* Clear status bit to acknowledge interrupt */
	ul_dummy = tc_get_status(TC0, 0);			//The compare bit is cleared by
reading the register, manual p. 915

	/* Avoid compiler warning */
	UNUSED(ul_dummy);

	ioport_set_pin_level(CHECK_PIN,HIGH);		//put test pin HIGH

	adc_start(ADC);
	while((adc_get_status(ADC) & 0x1<<24)==0);  //Wait until DRDY get high

	invalue=adc_get_latest_value(ADC);				//get input value


	for (uint32_t k=M;k>0;k--)
	{
		xbuff[k]=xbuff[k-1];
	}

	xbuff[0]=invalue;

	for (uint32_t x=0;x<M+1;x++)
	{
		value+=b[x] * xbuff[x];
	}

	outvalue = (value / 10000);

	dacc_write_conversion_data(DACC,outvalue);	//send output value to DAC

	ioport_set_pin_level(CHECK_PIN,LOW);		//put test pin LOW

}
```

Mät beräkningstiden för M=9.

Exekveringstid för M=9 (10 termer i summan): 6,06µs

Beräkna den ungefärliga beräkningstiden för en term.
Beräkning

I laboration 1 fick vi att beräkningstiden för A/D-D/A var 3,28 µs. Tar vi bort dessa 3,28 µs från de 6,06µs vi fick som exekveringstid nu borde det stämma överens med beräkningstiden för alla våra termer:

6,06-3,28 = 2,78 µs.

Vi har 10 termer, så den ungefärliga beräkningstiden för en term är $\frac{2,78}{10}$ *=278 ns*

Exekveringstid för en term: 278 ns

Jämför med beräkningstiden för flyttal ovan
Kommentar:

Tiden minskade drastiskt jämfört med tidigare tid när vi hade flyttalsberäkningar. Nu när vi har dem istället till uint32_t tar går det snabbare och därför ligger i god marginal.

Kommentar: Genom att i stället skala med en jämn multipel av 2, i detta fall t.ex. 8192, kan divisionen på slutet göras snabbare genom att utföras som ett antal skift åt höger.

Design av lågpass FIR-filter genom placering av nollställen

Vi skall nu se hur vi kan designa ett enkelt lågpassfilter med hjälp av placeringen av nollställena till överföringsfunktionen.

Överföringsfunktionen för FIR-filter ges av

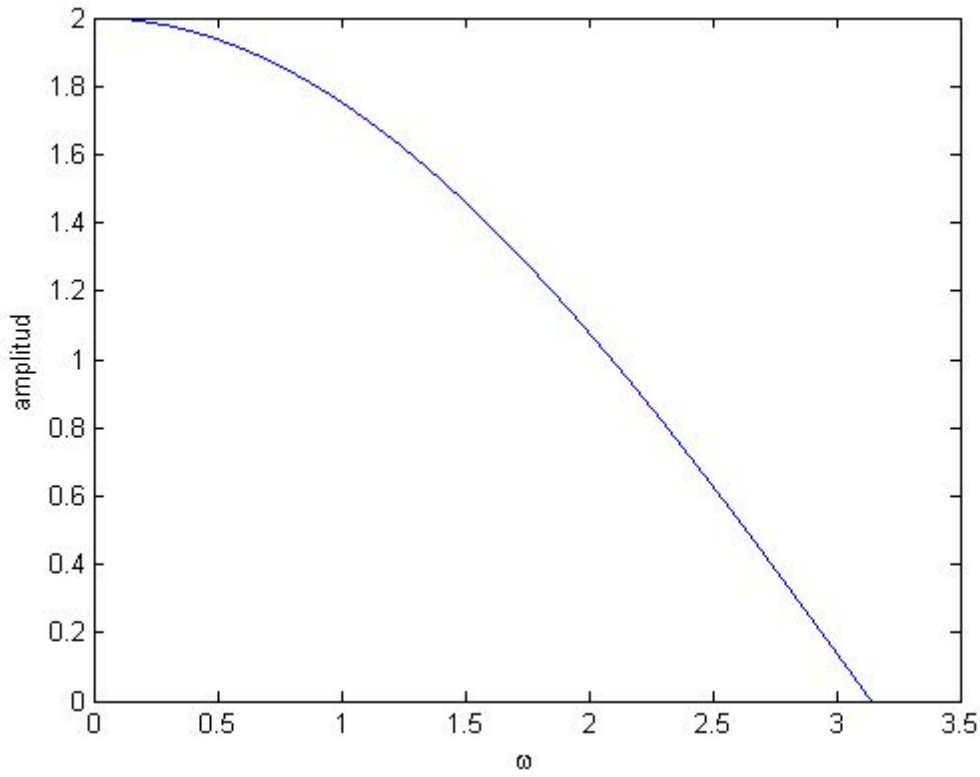
$$H(z) = \sum_{k=0}^{M} b_k \cdot z^{-k}$$

Låt oss börja med enklast tänkbara filter. Ett lågpassfilter borde spärra vid högsta användbara frekvens dvs. halva sampelfrekvensen som ju svarar mot den normerade vinkelfrekvensen $\hat{\omega} = \pi$. I komplexa talplanet svarar detta med $z = -1$. Vi skall således ha ett nollställe i denna punkt dvs. faktorn $(z+1)$ skall ingå i $H(z)$. Vi vet också att det för ett FIR-filter skall finnas lika många poler som nollställen och att alla poler ligger i $z = 0$.
Vi får således

$$H(z) = \frac{z+1}{z} = 1 + \frac{1}{z} = 1 + 1 \cdot z^{-1}$$

vilket innebär att $b_0 = b_1 = 1$.

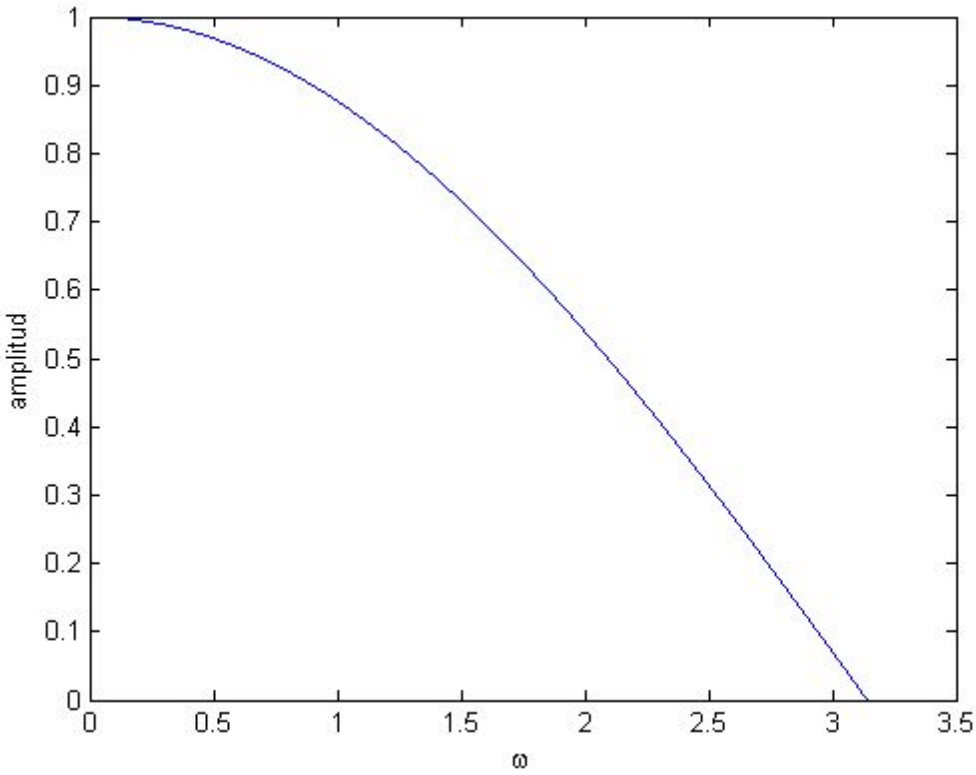
Använd Matlab, jämför med tidigare moment, för att plotta $|H(e^{j\hat{\omega}})|$ i intervallet $0 \le \hat{\omega} \le \pi$. (OBS variabeln: $\hat{\omega}$)
Klistra in plotten:



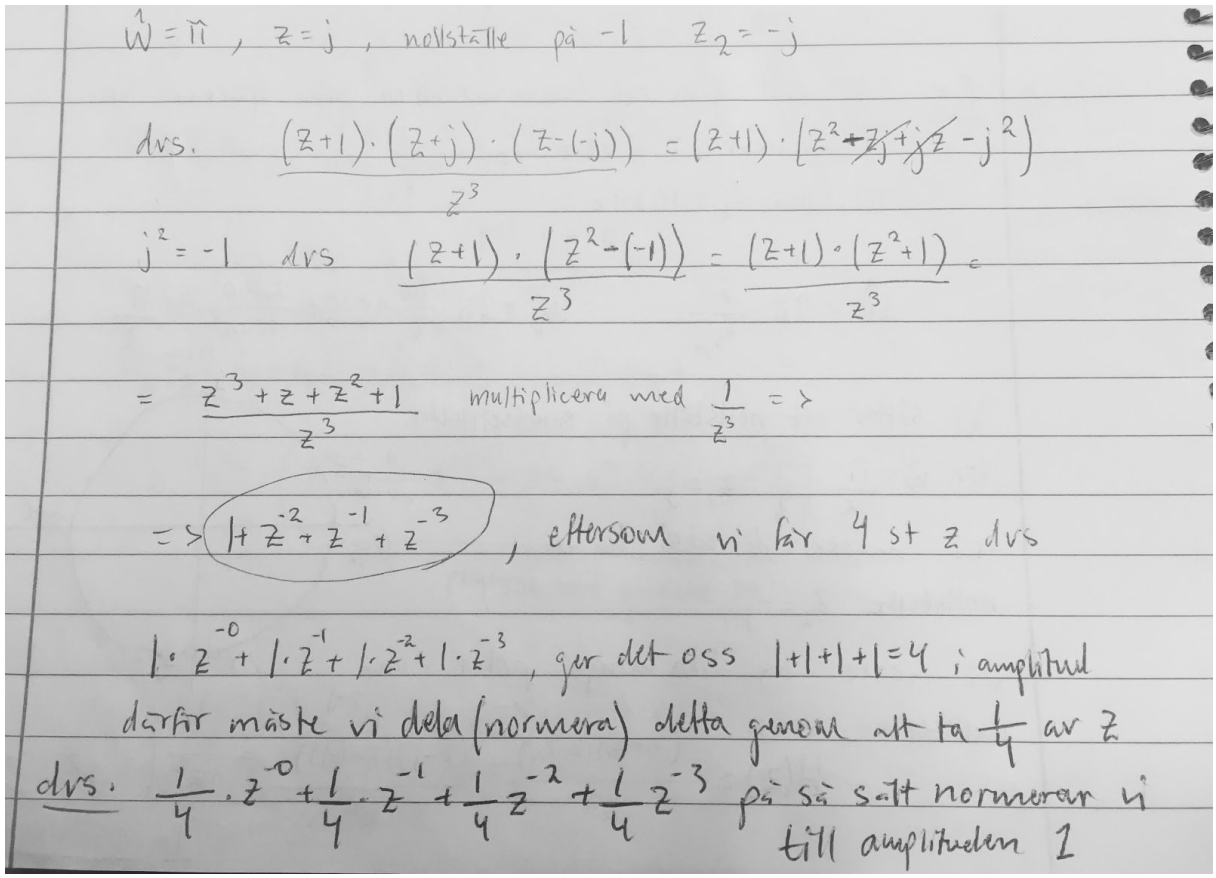
För ett lågpassfilter brukar man välja filterkoefficienterna så att $|H(e^{j\hat{\omega}})| = 1$ för frekvensen 0.
Om vi sätter in $\hat{\omega} = 0$ i uttrycket för $H(e^{j\hat{\omega}})$ ovan och bildar absolutbeloppet får vi

$$|H(e^{j\cdot 0})| = \left| \sum_{k=0}^{M} b_k \cdot e^{-j\cdot 0\cdot k} \right| = \left| \sum_{k=0}^{M} b_k \right|$$

Med $b_0 = b_1 = 1$ fås $\left|H(e^{j\cdot 0})\right| = 2$. För att normera gör vi således så att vi dividerar varje koefficient med 2. I detta fall får vi alltså $b_0 = b_1 = 0{,}5$
Gör en ny plot med Matlab med dessa värden.
Klistra in plotten:



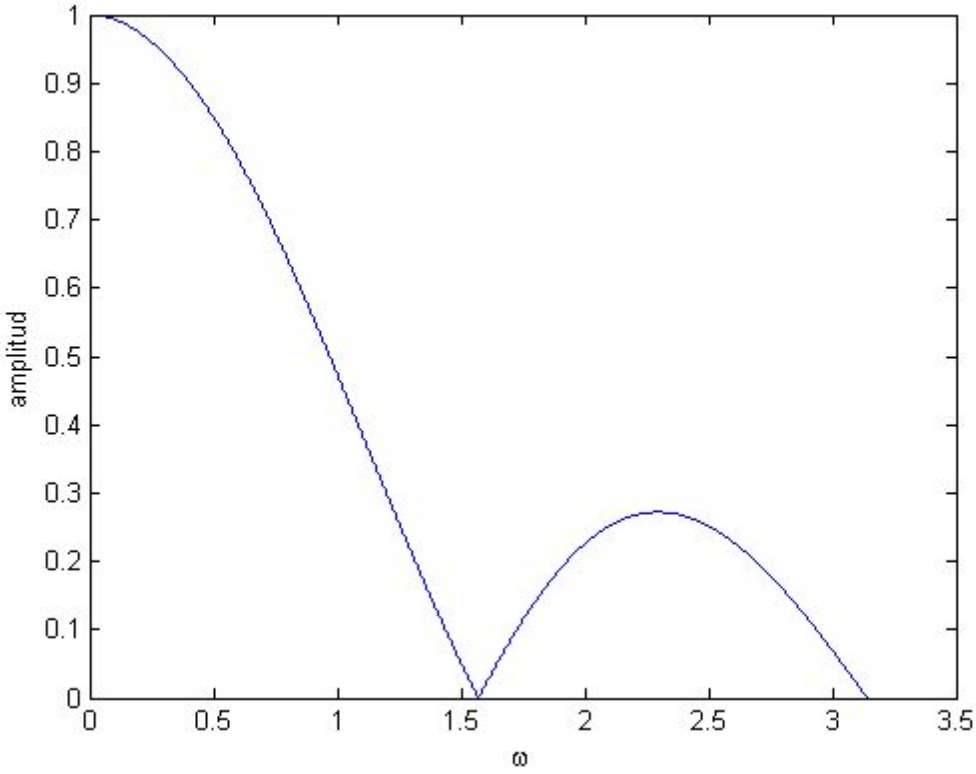
Vi går vidare genom att även placera ett nollställe i $z = j$ men eftersom komplexa nollställen alltid förekommer i komplexkonjugerade par måste det då även finnas ett nollställe i $z = -j$.
Bilda nu en överföringsfunktion med dessa tre nollställen och skriv den på en form så man kan identifiera filterkoefficienterna. Normera därefter så att $\left|H(e^{j\cdot 0})\right| = 1$.
Redovisa beräkningarna här:
(fota handskrivna beräkningar)



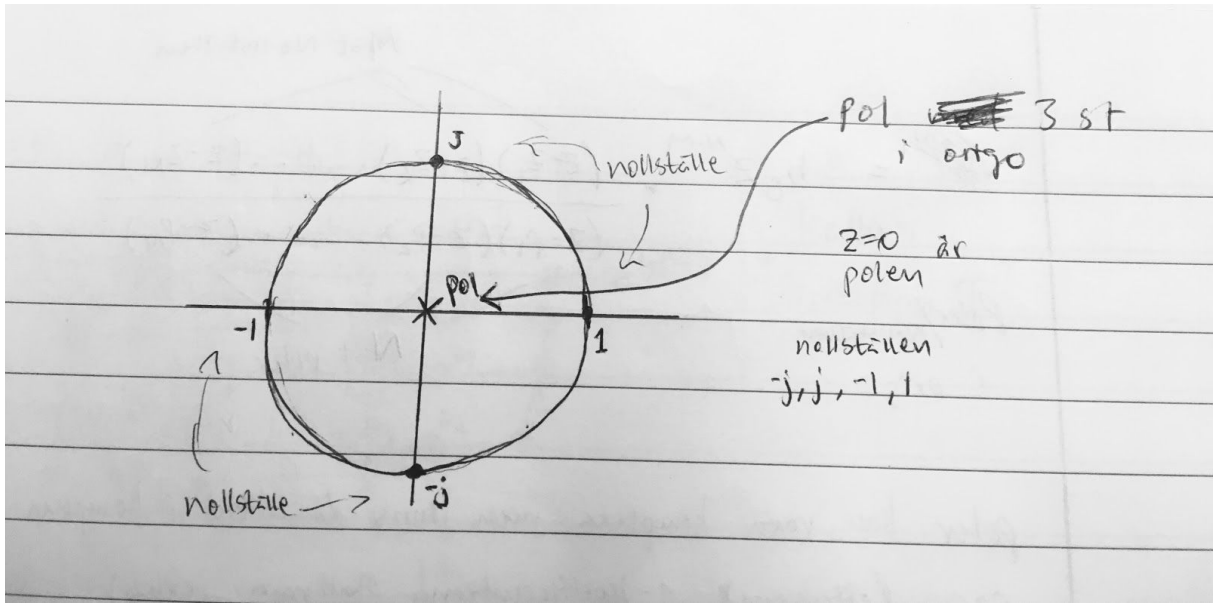
Vi får ett system med $1+z^{-1}+z^{-2}+z^{-3}$, dvs 3 grad och därför har 3 pol-nollställen i origo

Använd nu Matlab för att plotta $\left|H(e^{j\hat{\omega}})\right|$ i intervallet $0 \le \hat{\omega} \le \pi$.

Klistra in plotten:



Rita ett pol-nollställediagram för filtret, fota och klistra in:

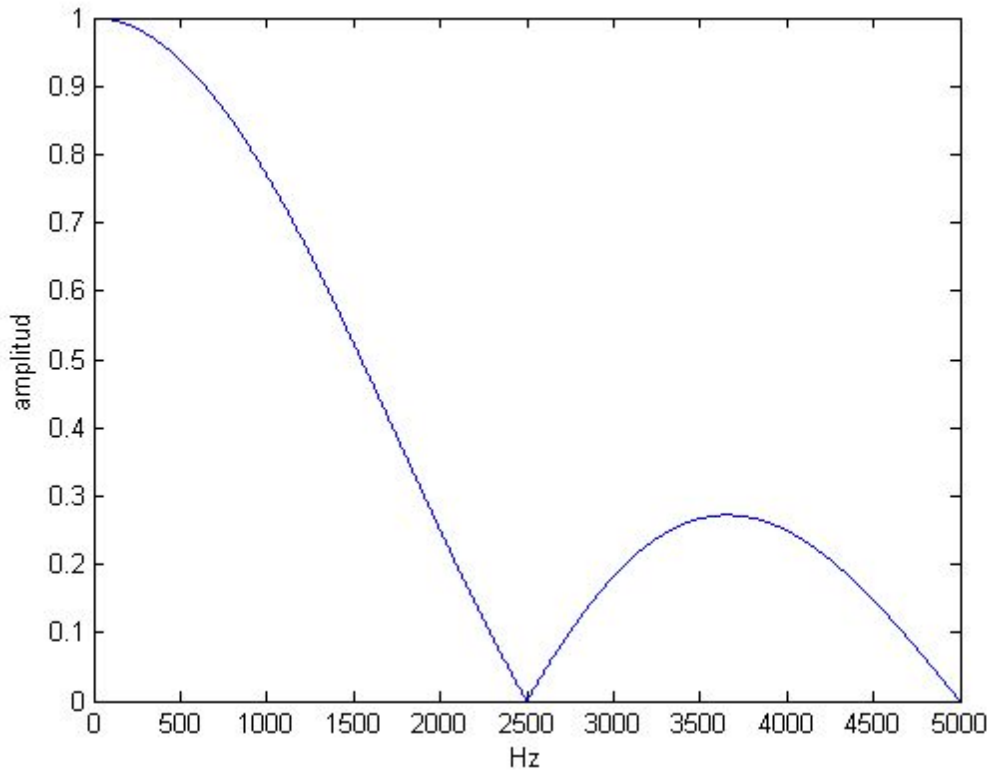


Vi får ett system med $1+z^{-1}+z^{-2}+z^{-3}$, dvs 3 grad och därför har 3 pol-nollställen i origo

Realisering och uppmätning av filtret

Nu vill vi testa om detta stämmer med verkligheten!
Men för att lättare kunna jämföra så gör om plotten ovan men nu med *f* som variabel.

Klistra in plotten:

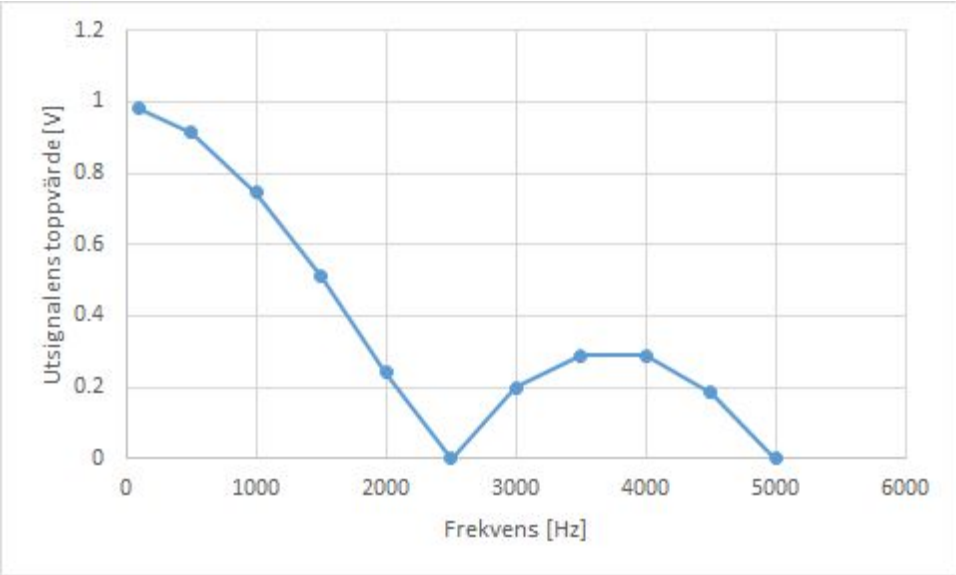


Använd samma uppkoppling som tidigare. Ställ in signalgeneratorn så att amplituden blir 1 V.
Använd FIR-filterkoden som du skrev tidigare. Enklast är att använda *float* varianten. Skriv in de nya *b*-koefficienterna med 4 decimaler.
Mät amplituden på utsignalen för ett antal frekvenser, börja med 100 Hz och gå upp till 5000 Hz på samma sätt som förut. Titta speciellt noga på de frekvenser som svarar mot nollställena.

Redovisa med tabell med mätvärden och ett diagram

Frekvens [Hz]	Utsignalens toppvärde [V]
100	0,982
500	0,915
1000	0,746
1500	0,51

2000	0,24
2500	0
3000	0,2
3500	0,29
4000	0,29
4500	0,185
5000	0



Kommentar:

Mätvärden från oscilloskopet stämmer ganska bra överens med plotten ifrån matlab ifall man kollar vissa punkter och jämför.

Använd filtret

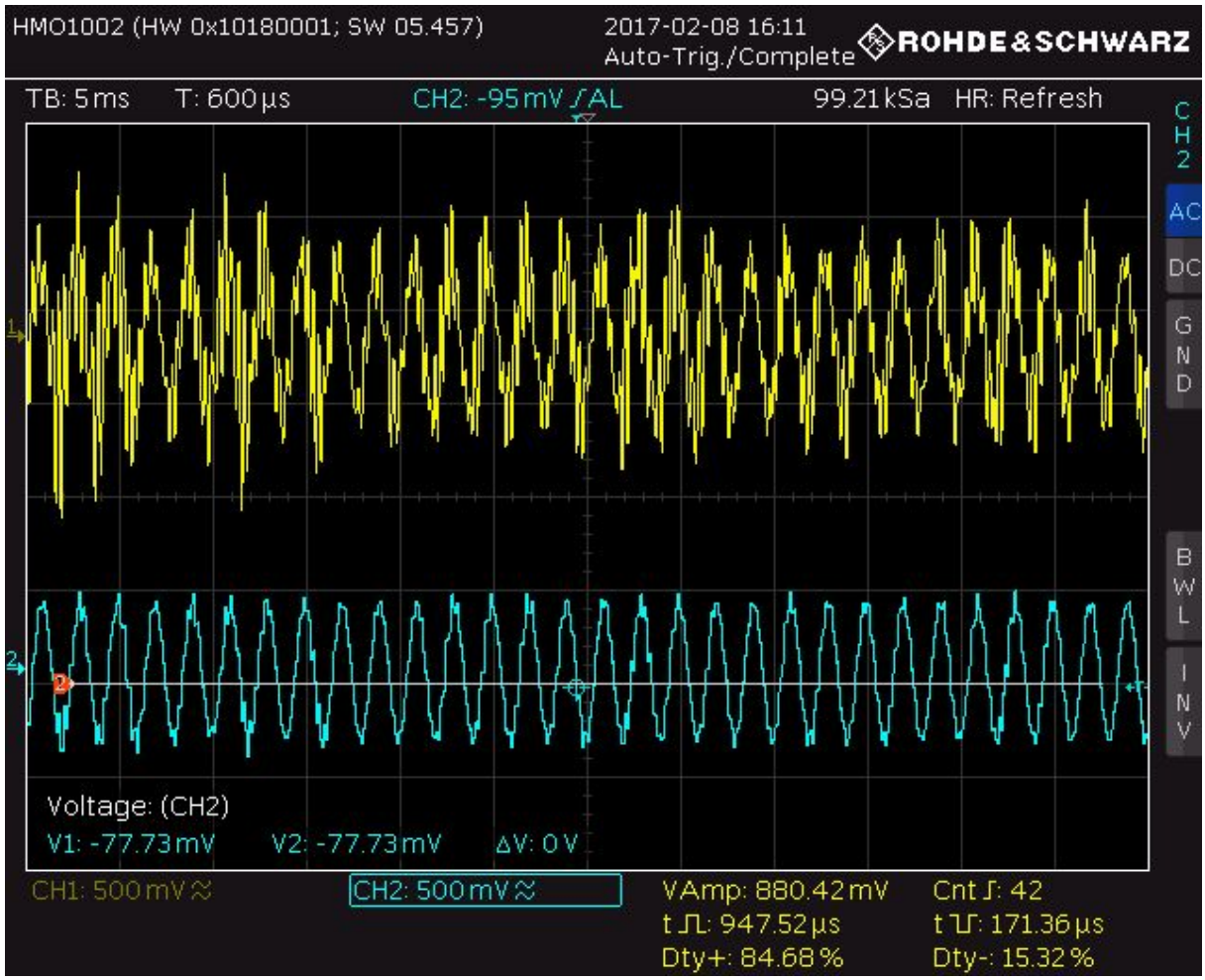
Till slut är dags att prova hur filtret kan minska på bruset i en signal. Ladda ner filen ”sin_noise.mp3” från kursens Its learningsida till din mobil. Den ligger i samma mapp som labhandledningen. Om du har en iphone kan du t.ex. använda appen ”MP3 Music Downloader Free”. Filen som är gjord i Matlab innehåller en sinussignal med ett överlagrat brus som ligger i frekvensområdet 2,5 – 4 kHz.

Anslut mobilen till ingången i stället för signalgeneratorn. Ställ in lämplig volym så du utnyttjar filtret maximalt. Studera insignal och utsignal med oscilloskopet.

Ange sinussignalens frekvens: 500 Hz

Fota oscilloskopskärmen med insignalen överst och utsignalen under.

Klistra in:



Kommentar:

Den filtrerade signalen är jämnare och signalen låter klarare, om än lite dämpad.

Gör om dokumentet till pdf och lämna in rapporten på Its learning. Var beredd på att kunna demonstrera programmet och förklara hur det fungerar.

Frivillig extra uppgift för den ambitiöse studenten…

Kan vi göra filtret ännu bättre? Vi går ett steg till och placerar även nollställen på enhetscirkeln svarande på $\hat{\omega} = \pm 3\pi / 4$. Detta svarar mot $z = -\frac{1}{\sqrt{2}} \pm j \cdot \frac{1}{\sqrt{2}}$.
Bilda nu som ovan en överföringsfunktion, nu med fem nollställen, och skriv den på en form så man kan identifiera filterkoefficienterna och normera så att $\left|H(e^{j\cdot 0})\right| = 1$.
Redovisa beräkningarna här (det blir en del… använd gärna lämpliga datorhjälpmedel, t.ex. WolframAlpha). Skriv avslutningsvis de erhållna b-koefficienterna med fyra decimaler:

Använd nu Matlab för att plotta $\left|H(e^{j\hat{\omega}})\right|$ i intervallet $0 \le \hat{\omega} \le \pi$.
Klistra in plotten:

Mät upp filtret på samma sätt som innan…

Testa med den brusiga insignalen, blev det bättre?

E