11/21/2014

Matlab & LEGO NXT

i anvendt matematikk og fysikk

Prosjekt i ING100 Ingeniørfaglig innføringsemne høsten 2014



|  |  |  |
| --- | --- | --- |
| Gruppenummer: | **1401** | |
| Medlemmer: | **Helge Johannessen Bjorland**  **Espen Ro Eliassen** | **228290**  **795934** |
| **Daniel Lovik** |  |
| **Anders Svalestad** |  |
|  |  |

Innhold

[1. Prosjektgjennomføringen 3](#_Toc403724827)

[2. Program 1: Legorobot (Utført av hele gruppen) 3](#_Toc403724828)

[2.1 Numerisk integrasjon (Utført av hele gruppen) 4](#_Toc403724829)

[2.2 Numerisk derivasjon (Utført av hele gruppen) 4](#_Toc403724830)

[2.3 Filtrering (Utført av hele gruppen) 5](#_Toc403724831)

[2.4 Kjøring av Legorobot (Utført av hele gruppen) 5](#_Toc403724832)

[3. Program plotTime: (Espen Ro Eliassen) 5](#_Toc403724833)

[4. Program cannonGame: (Espen Ro Eliassen) 9](#_Toc403724834)

[5. Program mathShow: (Espen Ro Eliassen) 14](#_Toc403724835)

[6. Program numJoy: (Espen Ro Eliassen) 19](#_Toc403724836)

[9. Konklusjon 20](#_Toc403724837)

[Referanser 20](#_Toc403724838)

[A. Matlabfiler 21](#_Toc403724839)

[A.1 Program 1 Obligatorisk del 21](#_Toc403724840)

[A.2 Program plotTime A.3 Program cannonGame A.4 Program mathShow 21](#_Toc403724841)

# Sammendrag

Skriv et sammendrag av hvordan programmene virker. Et sammendrag skal være kort og konkret, og skal ikke inneholde figurer eller kode. Det skal være en oppsummering av problemstilling, hvordan dere har løst problemet og hvordan det ferdige programmet fungerer (som en slags konklusjon). Dere kan gjerne strukturere sammendraget slik at hvert program beskrives hver for seg, for eksempel:

**Program1 (Hele gruppen i fellesskap):**

- hva som var problemstillingen

- hvordan dere har løst dette (detaljer om programmet)

- virker det etter hensikten?

**Sammendrag plotTime:**

Lage et program som illustrerer forskjellige måter å plotte oppdatere plot i matlab og hvor lang tid det tar. Måtene plot(x,y), set(plothandle, Ydata, Xdata) og refreshdata ble valgt. Alt presenteres først i en sanntids figure, og så en oppsummering. Det kommer klart frem at metoden en velger å plotte har stor betydning for hvor raskt programmet kjører.

**Program 3 (Utført av Kari og Anne):**

.

.

Skriv slik at leseren får lyst til å lese mer om programmene i selve rapporten.

# Prosjektgjennomføringen

Prosjektet startet med å avtale agenda for første gruppemøte og sette opp Github konto for å dele matlab kode med kildehåndtering. Første gruppemøte ble avholdt 16. September (2t). Ansvar ble fordelt og tisplan under ble vedtatt:

* 10 okt - ferdig med obligatorisk del
* 24 okt - Ferdig med skisse av kreativ del
* 7 nov - Ferdig med kreativ del slik at vi ferdigstiller rapport frem mot innleveringsfrist
* 21 nov - innleveringsfrist, prosjekt ferdig

Obligatorisk del ble gjennomført av alle gruppemedlemmene i hovedsak individuelt, men også noe i grupper alt etter hva som passet med arbeidssituasjon. Fokus var at alle gruppemedlemmer skulle få en god forståelse av den obligatoriske delen. Kommunikasjonsmåte mellom gruppemøter var via epost og telefon og roboten rullerte alt etter behov. Det tok ca 4 timer å bygge roboten (HJB) og hver person brukte omtrent 8 timer på obligatorisk del. dimer ﷽﷽﷽r:l følgende pl med. kle og middels enkle oppgaver og en vanskelig. Dette ble fordelt som følger:l følgende pl med.

10. oktober ble det avholdt nytt gruppemøte hvor vi ble enige om hvordan vi skulle kombinere den obligatoriske delen til en kode og fordelte kreative oppgaver å jobbe med. Vi ble enige om å gjøre flere enkle og middels enkle oppgaver og en vanskelig. Dette ble fordelt som følger:

Helge:

* Koordiner rapport
* Koordiner obligatorisk del i en fil
* Vis robotens hastighet under kjøring
* Initialiser joystick ved start
* Kreative oppgaver: 3, 7, 10, 22 og 26

Anders:

* Kreative oppgaver: 1 og 6

Espen:

* Kreative oppgaver: 23

Daniel:

* Kreative oppgaver: 5, 8 , 9 og 14

# 2. Program 1: Legorobot (Utført av hele gruppen)

Skriv og forklar hvordan funksjonene for integrering, derivering og filtrering fungerer. Få frem hvordan dette fungerer i forbindelse med Lego-roboten. Ta med ”screen shots” i forklaringen, utklipp fra koden og Matlabfigurer. Beskrivelsen skal fungere som en rask brukerveiledning slik at leseren skal kunne forstå hvordan dere har løst problemet. Husk å nevne eventuelle spesialiteter.

Helt kjapt så er vel mitt sammendrag slik:

1. Det er lagt innkode for å velge plotfrekvens, dette for å gjøre det mulig å kjøre uforstyrret av data plot som er ressurskrevende og dermed øker løkketiden i hovedløkken.

2. Først opprettes kobling mot NXT og Joystick, variabler og figurer initialiseres.

3. tiden siden oppstart legges inn i siste element i tidsvektoren

4. en beregner tiden på forige løkkegjennomkjøring ved å ta tiden i starten av denne løkken å trekker fra tiden i starten av forige løkke. Denne skal brukes som delta tid (x) i kalkulasjoner.

5. lyset leses inn og filtreres basert på siste lysverdig og siste filtrerte lysverdig.

6. avviket for lyset fra nullpunkt regnes ut (denne kunne gjerne ligget under les lys sensor)

7. avviket integreres og legges inn i en egen vektor, dette gjøres i en egen funksjon ved bruk av en delta tid og avviket fra nullpunkt som blir en slags f(t). en legger til sist verdi av integralet for å få hele integral summen.

8. tilsvarende integrasjon gjøres for en egen vektor, men denne bruker absolutt verdien av løkke integralet for dermed å få totalt avvik fra nullpunkt.

9. derivasjonen gjøres så i en egen funksjon ved bruk av en delta tid og delta y som regnes fra endringen i lyset verdi. (her ligger litt ekstra kode for å unngå ikke kalkulerbare verdier)

10. rettningen avgjøres i egen funksjon og legges i egen vektor, obs skulle være noe kode for at det må være en endring fra forige retning, dersom endringen gir 0 skal rettning beholdes lik den foregående.

11. verdien i konkurransen utregnes fra oppgitt formel og legges til slutten av en egen vektor.

12. joystickens verdier leses inn i egne vektorer og justeres for initiel nullstillingsverdi.

13. denne verdien kjøres gjennom et filter og oppdateres.

14. en bruker retningen til å sette variabler for visning av dette.

15. pådraget til hver av motorene regnes ut i egen funksjon basert på joystickens filtrerte innverdier og pådraget settes.

16. henter data fra hver av motorene for å avgjøre hastighet. dette gjøres ved å se på gjennomsnitts strekning motorene har beveget seg. som legges inn i egne vektorer ( dette kunne kanskje vært gjort til en egen funksjon)

17.plott frekvensen avgjør om det skal plottes (ved bruk av handles) eller om en bare skal oppdatere telleren som sjekker mot plott frekvens grensen.

18. dersom det plottes (se forslag til ny kode), plottes de 100 siste verdiene eller alle om det er ferre.

19. grafikk tvinges til å oppdateres

20. sjekker om koden skal avsluttes eller ny løkke skal kjøres, og prosessen startes eventuelt på nytt på 3.

21. når en ikke ønsker å kjøre lengre (løkken avsluttes) stoppes motorene. sensor koblinger stegnes og joystick slettes.

22. figurer fra kjøring stenges og nye figurer basert på all data som har blitt tatt vare på.

23. en løkke for å vente på at bruker skal kunne se på figurer settes før en kjøres til bruker trykker en knapp, da avsluttes programmet.

## 2.1 Numerisk integrasjon (Utført av hele gruppen)

Ta med kode og figurer som viser at dere har forstått hvordan numerisk integrasjon fungerer.

## 2.2 Numerisk derivasjon (Utført av hele gruppen)

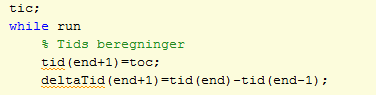
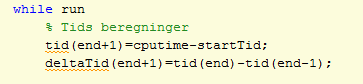
Ta med kode og figurer som viser at dere har forstått hvordan numerisk derivasjonen fungerer.

## 2.3 Filtrering (Utført av hele gruppen)

Ta med kode og figurer som viser hvordan filtreringen fungerer. Her kan dere inkludere mange forskjellige filtervarianter og vise at dere har testet og forstått filtrering. Knytt gjerne denne filtreringen opp mot resultatene fra numerisk integrasjon og derivasjon ved å vise hva som skjer dersom man filtrerer *før* integrasjonen/derivasjonen.

## 2.4 Kjøring av Legorobot (Utført av hele gruppen)

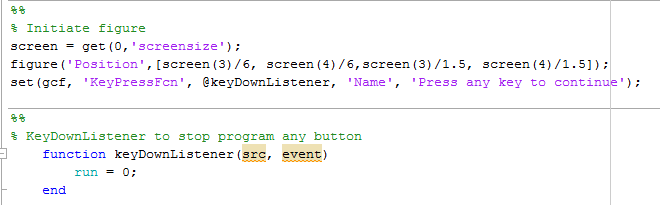
Ta med kode og figurer som viser hvordan manuell kjøring av Legoroboten fungerer totalt sett.



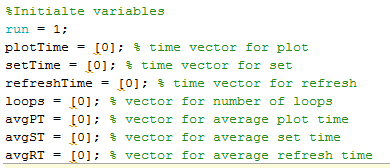
# 3. Program plotTime: (Espen Ro Eliassen)

Programmet er skrevet som en enkel funksjon. Det illustrerer tre forskjellige måter å utføre plotting i et program og presenterer disse i 3 underplott i en figur i sanntid. Plot 1 viser plot(x,y) funksjonen, plot 2 viser hvordan x- og ydata settes direkte via håndtaket til plottet og plot 3 viser oppdatering av plott via kildedata.

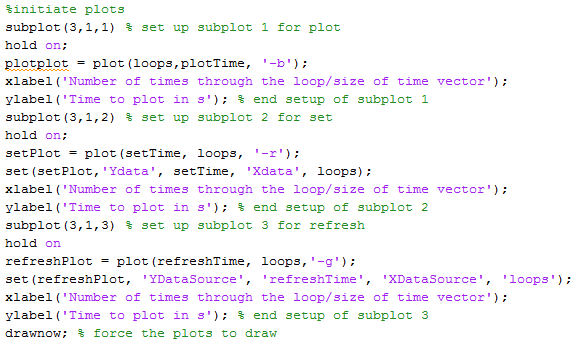
Det startes ved initialisering av figuren, som sørger for en grei plassering og størrelse ut fra skjermens størrelse. Det registeres også en “keyDownListener” som figurens “KeyPressFcn”. Denne defineres som en under funksjon og skal brukes for å gå videre i programmet.



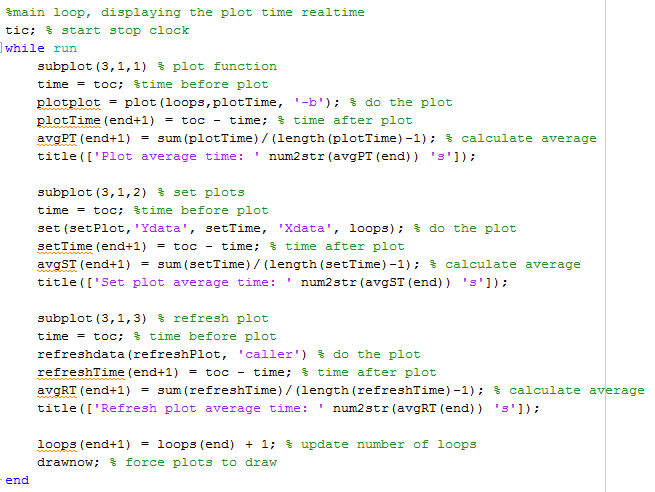
Deretter initialiseres programspesifikke data. Dette er vektorer for tid og gjennomsnitts tid for de forskellige måtene å plotte på. En vektor for antall ganger gjennom hoved løkka og variabel for styring av hoved løkka.



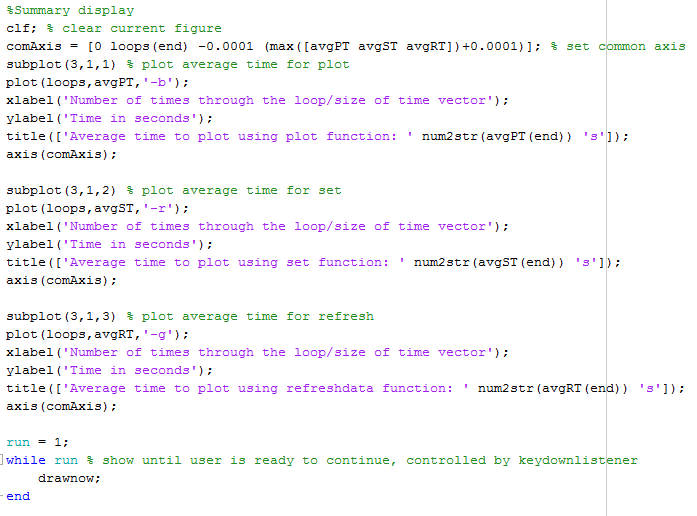
Neste steg er å initialisere plottene definere egne håndtak til hvert av plottene og gi aksene tekst. Plot(x,y) funksjonen gis også et håndtak, selv om det ikke benyttes til noe. Avsluttende i denne seksjonen tvinger en programmet til å tegne opp plottene.



Hoved løkka i programmet er neste seksjon, og stoppe klokka startes rett før en går inn i løkka. Deretter går en inn i hvert underplott tar tiden rett før og rett etter hvert plot og lagrer det i den tilsvarende vektoren. En regner så gjennomsnittstid for den bestemte typen plott og viser den gjennomsnittstiden som gjelder for denne gjennomkjøringen i tittelen til plottet. Dette gjøres for alle tre plott. Rett før løkka avsluttes oppdaterer en hvor mange ganger løkken har blitt kjørt gjennom og tvinger grafikken til å oppdatere seg. Løkka avsluttes ved å trykke en hvilkårlig knapp.



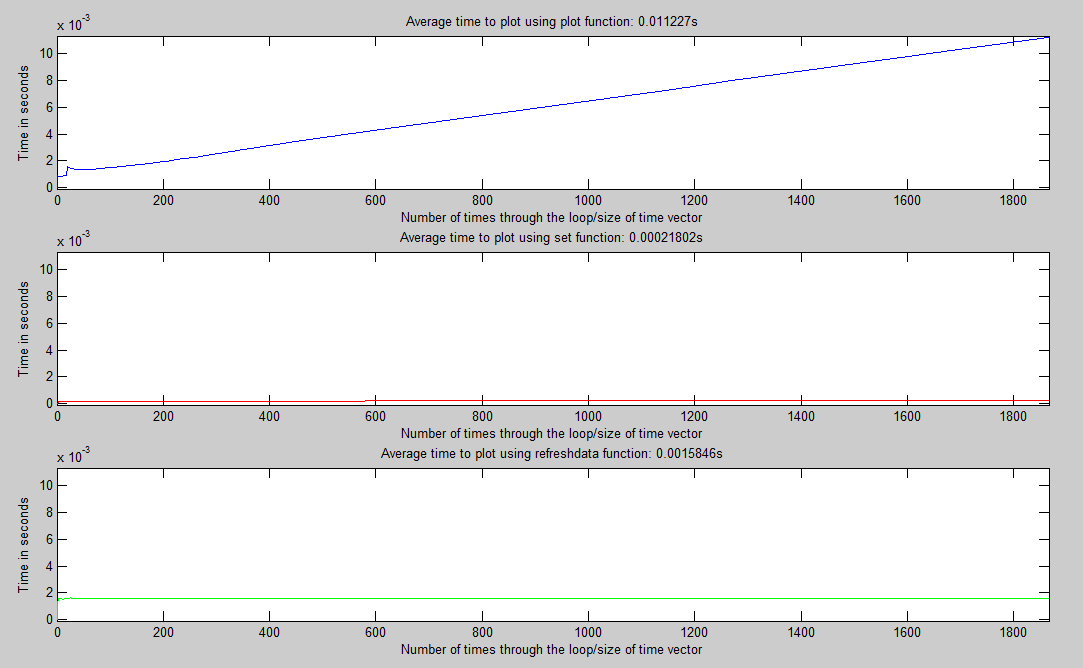
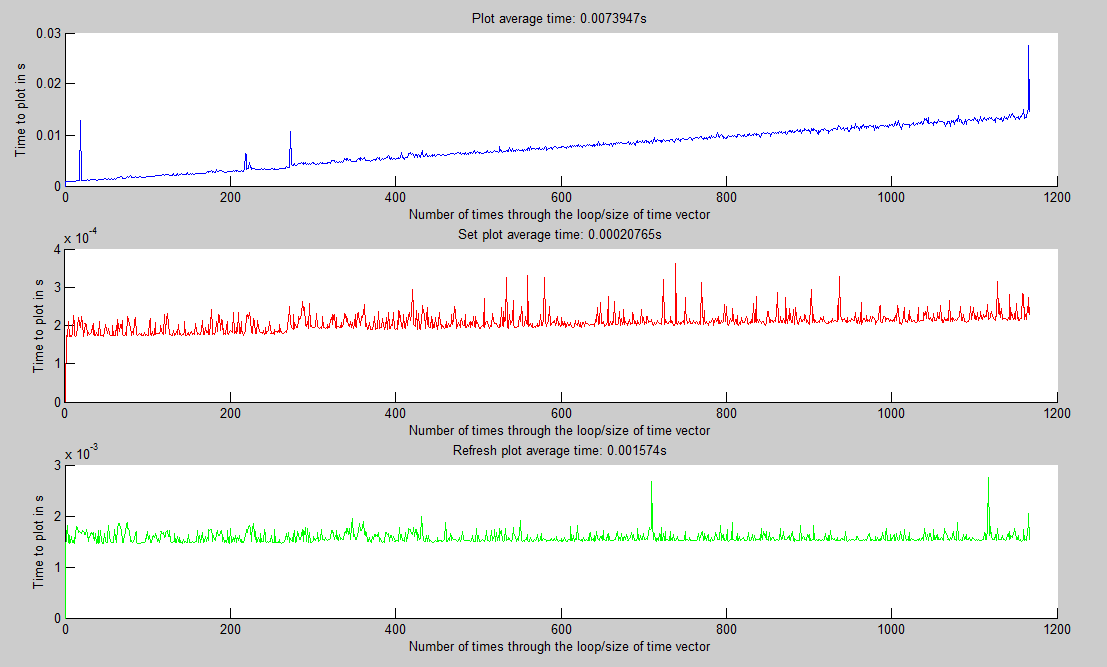
Når en er ferdig å se sanntidspresentasjonen skal programmet vise en oppsummerings figur. Nåværende figur renses og det defineres felles aksegrenser for alle 3 plottene. I dette oppsummeringsvinduet vises grafen for gjennomsnittstidene til de forskjellige plott metodene i tilsvarende underplott. Dette gjøres alle ved bruk av plot(x,y) funksjonen da målingene nå er over. Til slutt lar en programmet gå i en løkke som tvinger grafikk oppdatering frem til bruker ønsker å gå videre ved å trykke hvilken som helst knapp.



Når programmet anses som ferdig stenger en nåværende figur.

**Konklusjon plotTime:**

Det kommer helt klart å tydelig frem at å sette x- og ydata direkte via håndtak i et oppdaterende plott er minst resurskrevende/tar kortest tid. Oppdatering av x- og ydatakilde fungerer også bra, mens å bruke plot(x,y) funksjonen om og om igjen blir mer og mer ressurskrevende jo større vektorene blir.

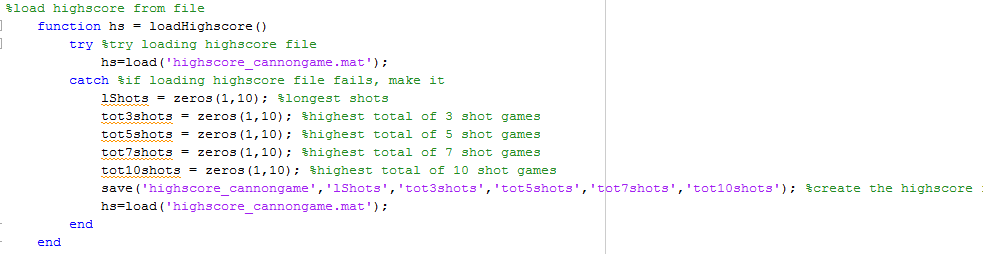


# 4. Program cannonGame: (Espen Ro Eliassen)

Programmet er skrevet i en funksjon som igjen inneholder under funksjoner. Det startes med å definere konstante verdier som skal brukes i koden. Blant disse finner vi skjermstørrelse, et forhold mellom skjermstørrelse og plot størrelse, plott akser, gravitasjonsens akselerasjon og start fart til prosjektilet. Se seksjon “Constants”.

Etter konstanter følger initialisering av variabler. Disse inneholder både betingelses variabler for å vise forskjellige deler av programmet og variabler for mattematiske kalkulasjoner. Poengsum liste lastes her inn fra en underfunksjon “loadHighscore()”. Se ellers seksjon “Initiate variable”.

loadHighscore() funksjonen er en underfunksjon som returnerer en struct variabel. Denne hentes inn fra filen “highscore\_cannongame.mat” og inneholder feltene lengste skuud (“lShots”), lengste total på 3, 5, 7 og 10 skudd, (“tot3shots”),(“tot5shots”), (“tot7shots”) og (“tot10shots”). Om ikke filen finnes skal den lages. Hvert felt inneholder en 1x10 matrise.

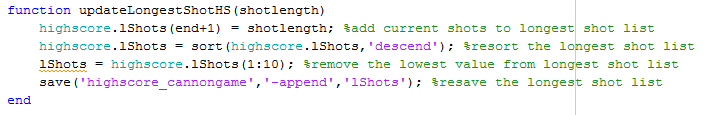


Når konstanter og variabler er satt opp er det på tide å starte selve spillet. Det gjøres først ved noen begynnende valg, se “Initial game options”. Her har en mulighet for å velge en- eller flerspiller, se høyeste poengsummer eller gå ut av spillet. Alt styrt ved bruk av menu funksjonen som gui. Etter en har valgt antall spillere får en mulighet å velge vanskelighets grad. Vanskelighetsgraden avgjør hvor mange radianer det er fra en vist vinkel til den neste. Altså utskytningsvinkelen endrer seg fortere og det blir vanskeligere å treffe den optimale vinkelen.

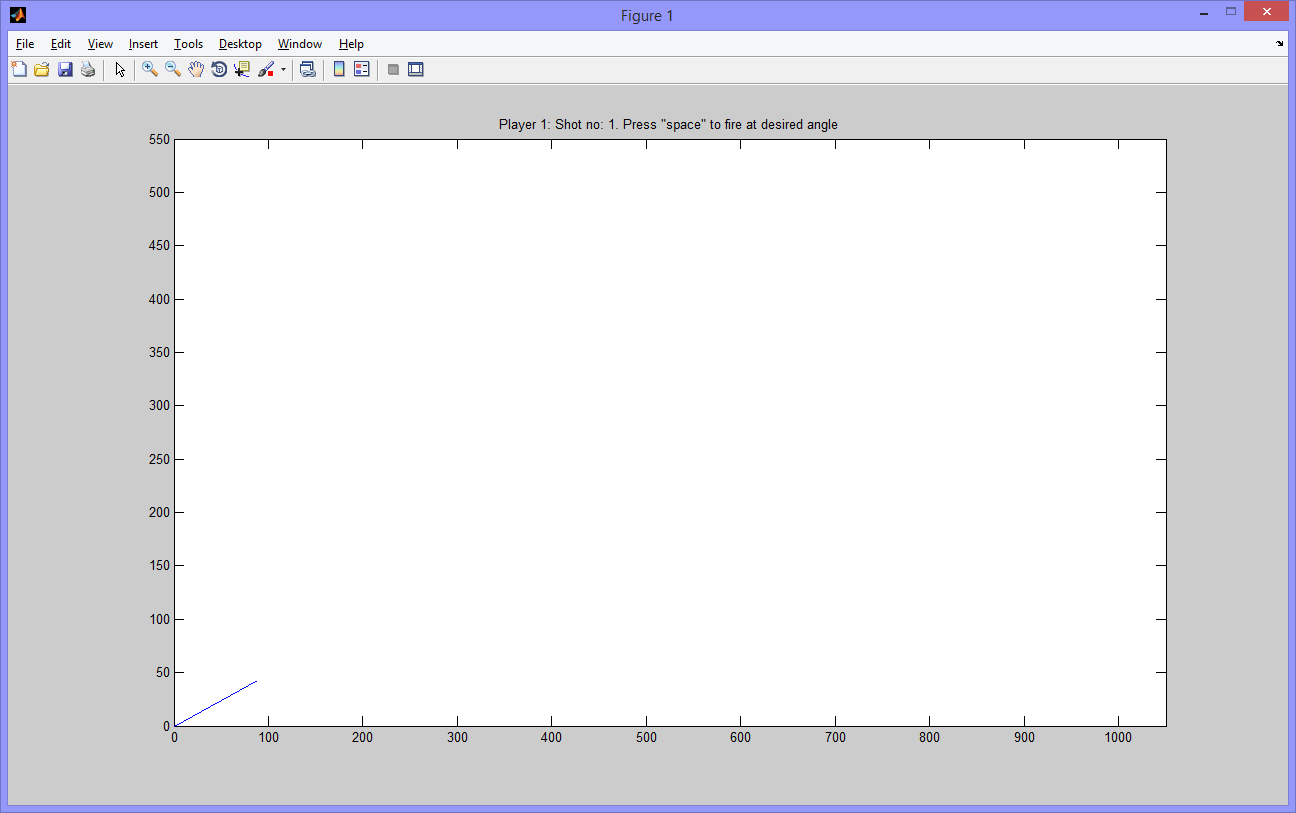
Hovedfiguren defineres i en egen seksjon, se “Initialize figure”, dynamisk ut fra skjermstørrelsen. Den starter 1/6 av høyde og bredde fra hjørnet og strekkers seg høyde og bredde delt på 1.5 (2/3 av de respektive størrelsene) fra start punktet. Figurens “KeyPressFcn” defineres som underfunksjonen “keyDownListener”. I tillegg så settes figuren slik at den ikke kan forandres størrelse på. Det kan nevnes at det kunne vært satt flere under funksjoner. F.eks ville en funksjon som håndterte figurens “CloseRequestFcn”, være på sin plass i et reelt spill.

Underfunksjonen “keyDownListener” inneholder 2 muligheter. Ved at bruker trykker mellomrom setter den variablene “space” og “showangle” til 1 og 0. Denne vippen brukes for å kontrollere fremgang i selve hoved løkken. Når escape trykkes settes “stop” variablen til 1.

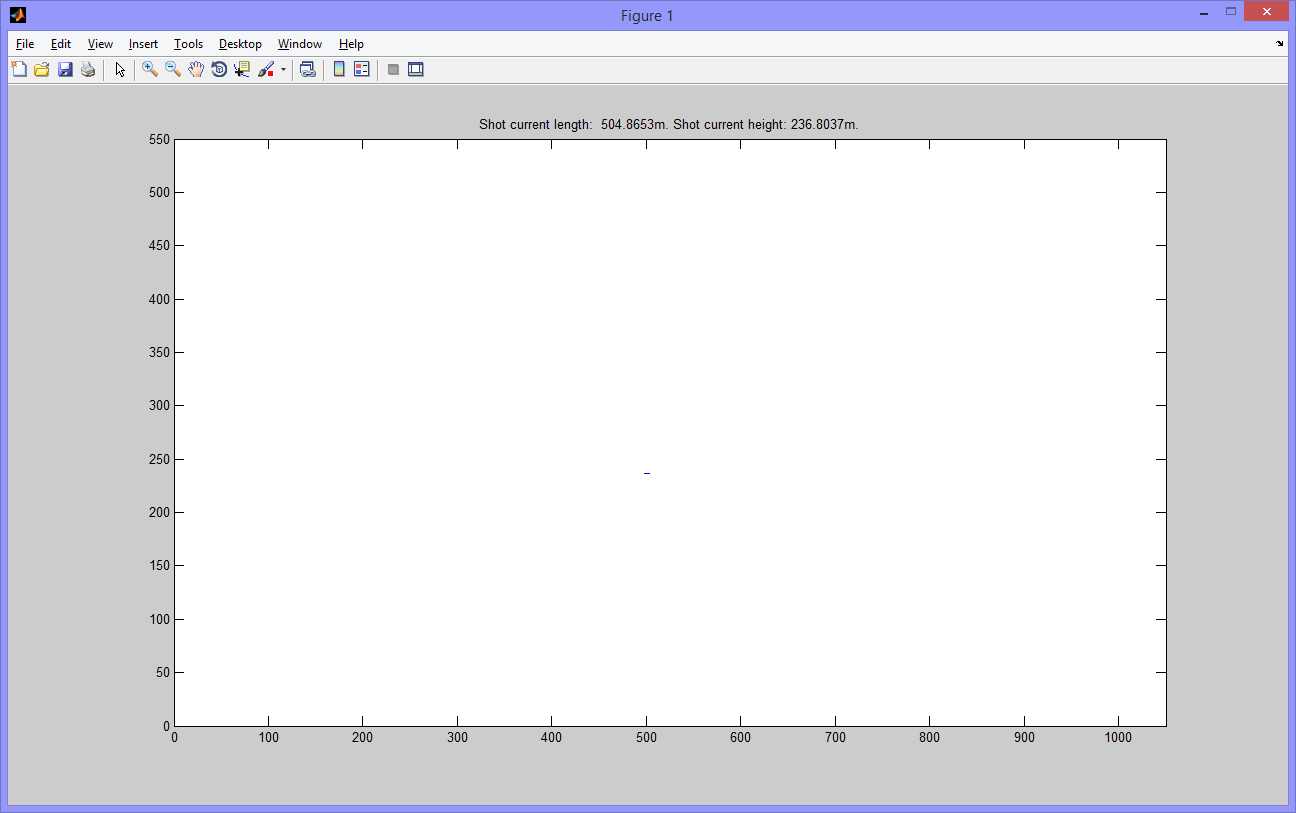
Før selve hovedprogrammet/-løkken kommer det flere underfunksjoner som har forskjellige oppgaver. De to neste på listen er updateLongestShotHS og updateTotalLenghtHS. Disse funksjonene svært lik funksjonalitet. Den første oppdaterer lengste skudd listen, mens den siste oppdatere den riktige høyeste totaltsum listen basert på hvor mange runder er valgt. Grunnen til at disse er skilt i 2 funksjoner er fordi lengste skudd skal sjekkes/oppdateres hvert skudd, mens total sum oppdateres ved endt spill. Logikken for oppdatering er lik i begge funksjonene. Det gjøres ved at oppnådd lengde/sum legges til listen, så sorteres den og det siste elementet fjernes.



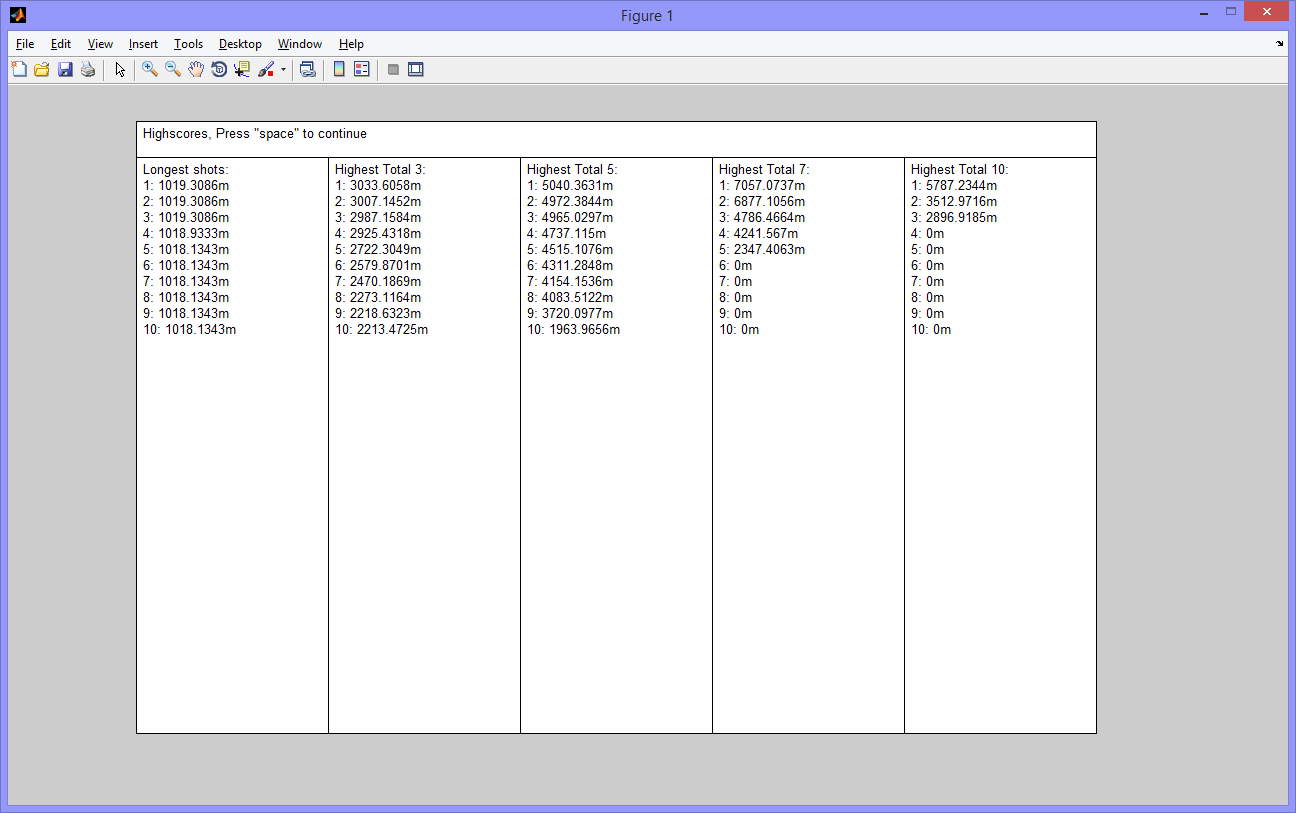
showAngle er en av de mer spennende funksjonene i programmet. Det er en funksjon som styrer direkte i hoved funksjonaliteten til spillet. Den viser en linje som først øker fra 0° mot 90° grader for så å gå ned igjen mot 0°. Plaseringen til denne linjen når spiller trykker mellomrom knappen angir vinkelen prosjektilet skytes ut med. Linjen den tegner som angir vinkelen gis av et x,y punkt som regnes med trigonometriske funksjoner og origo. Her benyttes også konstanten chartratio på y kalkulasjonen for å gi en illusjon av at linjen er tilnærmet like lang hele tiden, selv om skalaen på x og y aksene er forskjellige. Det er ingen returverdi fra denne funksjonen, istedenfor setter den angle variablen direkte. Noe som er fordelen med å ha den som en egen underfunksjon, fremfor å trekke den ut som en enkeltstående funksjon. Se kode i appendiks og figur under.



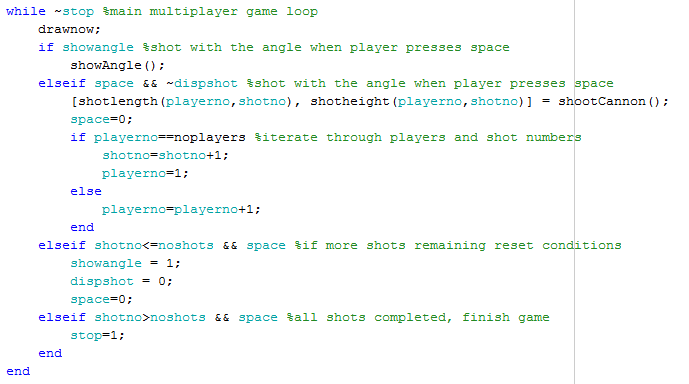
Når en vinkel for skuddet er angitt er det under funksjonen shootCannon() som tar seg av både kalkulering av skudd og animasjon. Selve kalkuleringen gjøres ved å dekomponere utgangshastigheten, gitt av konstanten “initialSpeed”, i x og y deler ved vinkelen. Deretter kalkuleres det hvor lang tid det vil ta før prosjektilet treffer bakken igjen. Ut i fra denne kalkulasjonen lages en tidsvektor som igjen brukes til å animere selve skuddet. Det er vært å legge merke til at en antar at prosjektilet starter på bakken altså y=0. Lengden på skudded regnes altså ut fra dekomponert x fart og tid i luften. Utregningen stemmer også overens med formelen for strekning gitt av . Dette kan bevises ved å manipulere koden til å kjøre med vinkel 45° som gir det lengste skuddet. Verdt å merke seg er at prosjektilet animeres via to og to punkter i flyvebanen. Det hadde også vært mulig å gjøre det punktvis. Funksjonen returnerer både høyde og lengde på skuddet, selv om det bare er lengde som tas med i poengsummer. Det er fra denne funksjonene at lengste skudd oppdateringen for top listen oppdateres.



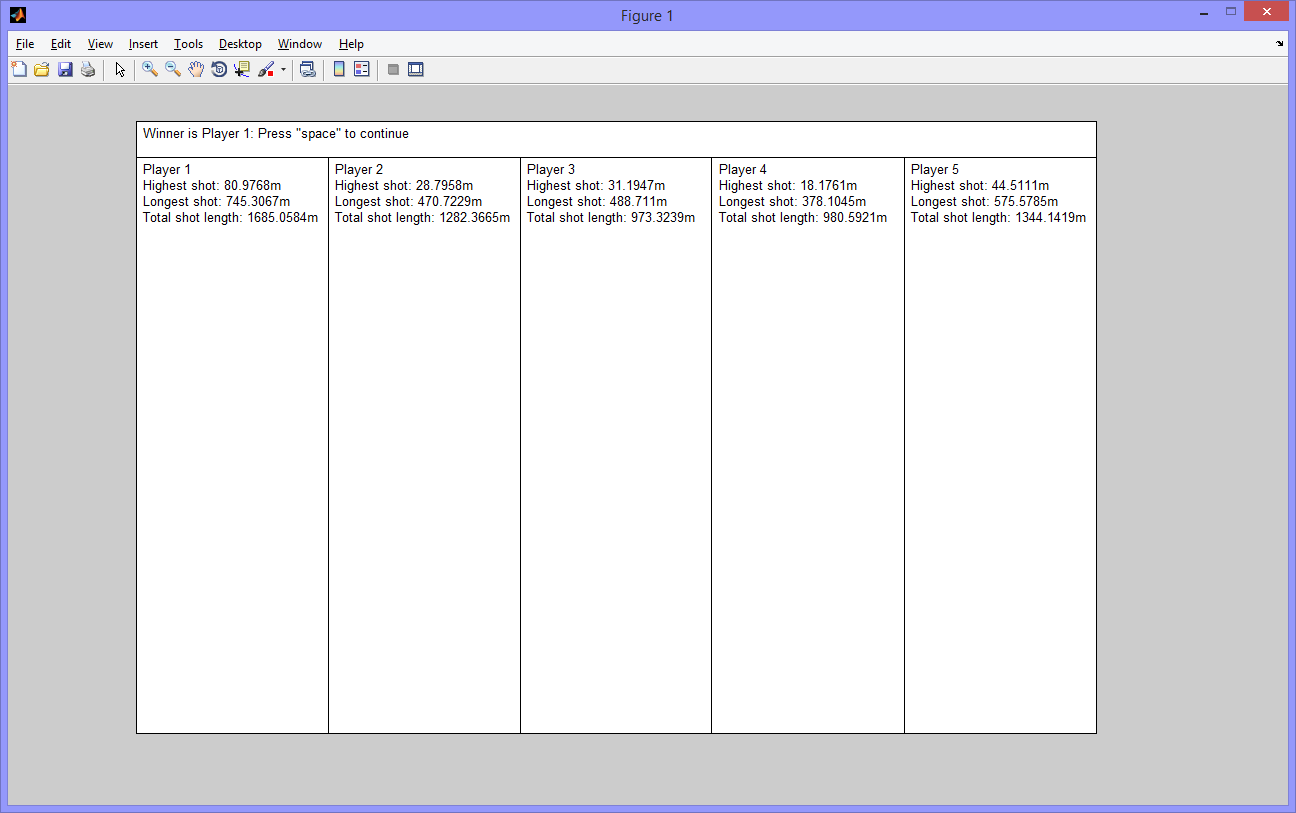
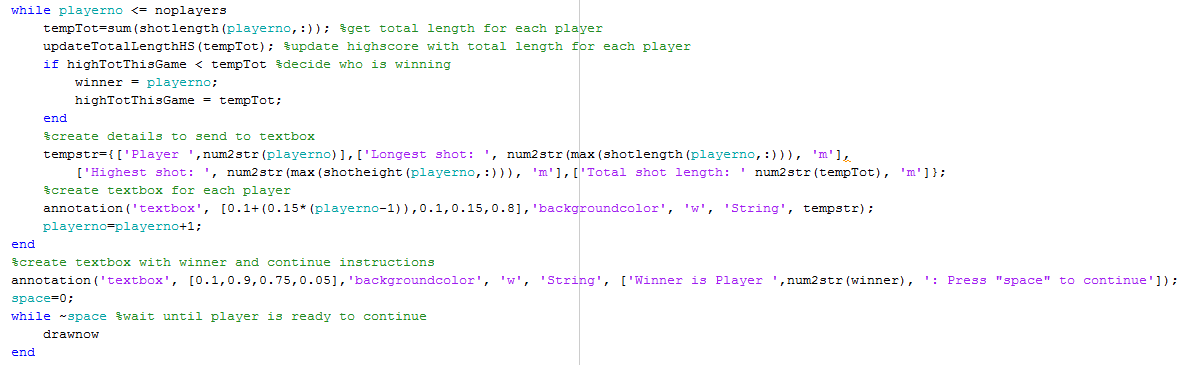
Siste under funksjon er showHighscores(), som rett og slett kalles fra de innledende valgene. Den viser alle høyeste poengsum / lengste lengde listene. Logikkmessig er det lite spennende som skjer her. Det ligger noe enkel bearbeiding av variabler for å få det på riktig format for å vise dem med annotation funksjonen. Deretter kjøres en løkke for å vente til spiller er klar til å gå videre.



Selve hovedløkken til programmet er egentlig 2 svært like løkker. Det er en for enspiller og en for flerspiller. Forskjellen ligger i noe ekstra logikk for å håndtere at det flere spillere. Derfor er det kun flerspiller løkken som beskrives her. Det settes opp matriser for å holde rede på skuddlengde (og høyde), hvor hver rad i i matrisen representerer en spiller og hver kolonne et skudd. En starter med spiller 1 tegner opp figuren og begyner med å kjøre showAngle(). Når spiller har trykket på på mellomrom for å skyte, flippes variablene via keyDownListener og space blir 1 og showangle 0. Dette plukkes opp i en if/elseif/else logikk. Dette sørger for at skuddet vises frem til spiller er klart til å gå videre og trykker mellomrom igjen. Dersom det er flere skudd igjen settes showangle variablen til 1 igjen og hele prosessen gjentas. Dette gjøres om og om igjen til alle spillere har skutt alle sine skudd (eller escape knappen blir trykket). Da settes stop variabelen og løkken avsluttes.



Deretter kjøres en løkke som går igjennom summene til alle spillerne, avgjør hvem som vant og setter opp spill detaljer for hver spiller. Når alle spillerne er blitt vurdert settes en egen boks over detaljene med hvem som vant, og koden går inn i en løkke som venter på at spillere er klar til å fortsette.



Avslutningsvis ligger det ligger det en liten logikk som kaller selve funksjonen selv igjen for å la spillere spille på nytt. Sett fra et minneperspektiv er det ikke noen ideel situasjon og kalle opp funksjonen inne i seg selv. Men i dette tilfellet anses muligheten for at noen klarer å sitte å spille sammenhengende nok til å klar å bruke opp minne på en moderne maskin som svært usannsynlig. Et alernativ hadde vært å flytte seksjonen “Initial game options” ut i en egen gui funksjon som kontrollerte dette. Dersom exit blir valgt i menyen, så vil programkoden kjøres til ende og programmet / funksjonen er helt avsluttet.

**Konklusjon cannonGame:**

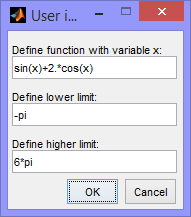
Programmet er skrevet som et spill og derfor er noe av logikken tilpasset at det skal være et spill. F.eks er animeringstiden av skuddene økt slik at spillere skal få en følelse av at de skyter noe virkelig. Samtidig er mange hensyns som en ville tatt i design av et slikt prosjekt utelatt, både av tids og relevans messige årsaker. Logikken i form av dekomponering av hastigheter gir samme resultat som regneregler for strekning.

# 5. Program mathShow: (Espen Ro Eliassen)

Funksjonen mathShowMain er et initielt brukergrensesnitt for å velge om en ønsker å derivere eller integrere. Den ligger i en løkke og kaller hver av det spesifikke funksjonene for seg, og noe som gjør at disse kan avslutte når de er ferdig istedenfor å kalle seg selv for å holde programmet gående (denne metoden ble vist i cannonGame programmet). Se siste paragraf om cannonGame programmet.

mathIntShow-funksjonen viser nummerisk integrasjon med 5 plot. 1: Selve funksjonsgrafen, 2: den nåværende løkkens bidrag til integralet, 3: integralet fra nedre grense til den nåværende løkken, 4: integralet for hele det definerte området og 5: grafen for funksjonen til integralet.

Funksjonen starter med at bruker angir funksjon, nedre og øvre grense. Dette forutsetter at bruker vet hvordan en skriver matematiske funksjoner i matlab. Inndata fra bruker lagres så til 3 variabler som strenger. Det er viktig at brukeren benytter x som variabel i funksjonen.



Deretter settes figuren dynamsik ut fra skjermstørrelse og en keyDownListener funksjon registeres som figurens “KeyPressFcn”. Se cannongame for mer om dette. Instrukser for å gå videre i programmet settes i figur navn, trykk hvilken som helst knapp for å få videre. Hele denne initialiseringen er gjort i en underfunksjon.

Forutenom kallet av intiialiserings funksjonen for figuren er hele hovedprogrammet satt i en try/catch logikk. Dette fordi programmet er ikke satt opp til å håndtere komplekse tall eller grenser på 0, og dette lar programmet unngå å krasje helt.

Variablene som kommer inn fra bruker er på streng form, for grensene må denne konverteres til tall før de brukes videre. Deretter sjekkes det at øvre og nedre grenser er definert riktig i forhold til hverandre, om ikke så rettes dette.

Det lages en initiel x vektor fra nedre til øvre grense som øker med 0.001 pr x. Om dette gir en vektor med mer enn 10000 elementer økes dette hoppet fra pr x til vektoren har fått en størrelse som er 10000 elementer eller mindre. Deretter evalueres strengen som bruker har angitt som funksjon for å skape en y vektor basert på x vektoren. Felles grense for aksene på plottene defineres og integralet summeres for hele det definerte området.

Underplott 1 og 4 er faste hele tiden og settes nå opp ved bruk av plot(x,y) og bar(x,y) funksjonene. Restene av plottene initialiseres så langt de kan.

Før hovedløkken kjøres i gang settes x til å kun inneholde nedre grense som verdi. Det definers en maxTime variabel som er hvor mange sekunder hele området skal være. En teller i, samt 0 verdier i andre variabler/vektorer som skal benyttes. Til slutt startes stoppeklokken.

Inne i selve hovedløkken leses først stoppeklokken av. Så lenge denne er under angitt maks tid, så legges den til som en ny verdi på slutten av x vektoren. Ved første gjennomkjøring av løkken vil alle vektorer ha 2 elementer dette er viktig for å kunne tegne rektanglene som integralene utgjør.

For hver gang gjennom løkken evalueres innstrengen fra burker for å danne en y vektor som stemmer overens med x vektoren. Siden koden alltid vil ta litt tid å kjøre gjennom vil det alltid være en endring i tid og det er denne endringen som benyttes i nummerisk integrasjon. Det er ønskelig at denne endringen skal bli så liten som mulig.

Rektanglene som integralet utgjør tegnes ved å bruke fill funksjonen. Her skal linjene som lages av x,y punktene utgjøre et rektangel,alstå må det være 4 punkter. For X blir dette forige x, denne x, denne x og forige x. Mens for y blir det 0, 0, y nå og y nå. Funksjonen til integralet beregnes ved å legge til summen av den nåværende løkkens integral til summen av integralet så langt.

Når tiden overstiger området nullstilles de grafene som skal det, og x verdien settes til nedre grense igjen. Til slutt resetes stoppeklokken.

Selve hovedløkken avsluttes med å tvinge grafikk oppdatering.

Programmet fanger feil som produseres av logikken i inne i try. Da viser den en feilmelding om feil med funksjon eller grenser. Dette er bare en standard feilmelding, og ikke en nøyaktig tilbakemelding på hva som gikk galt. Det lar bruker tenke et sekund så avsluter den og går tilbake til hoved brukergrensesnitt.

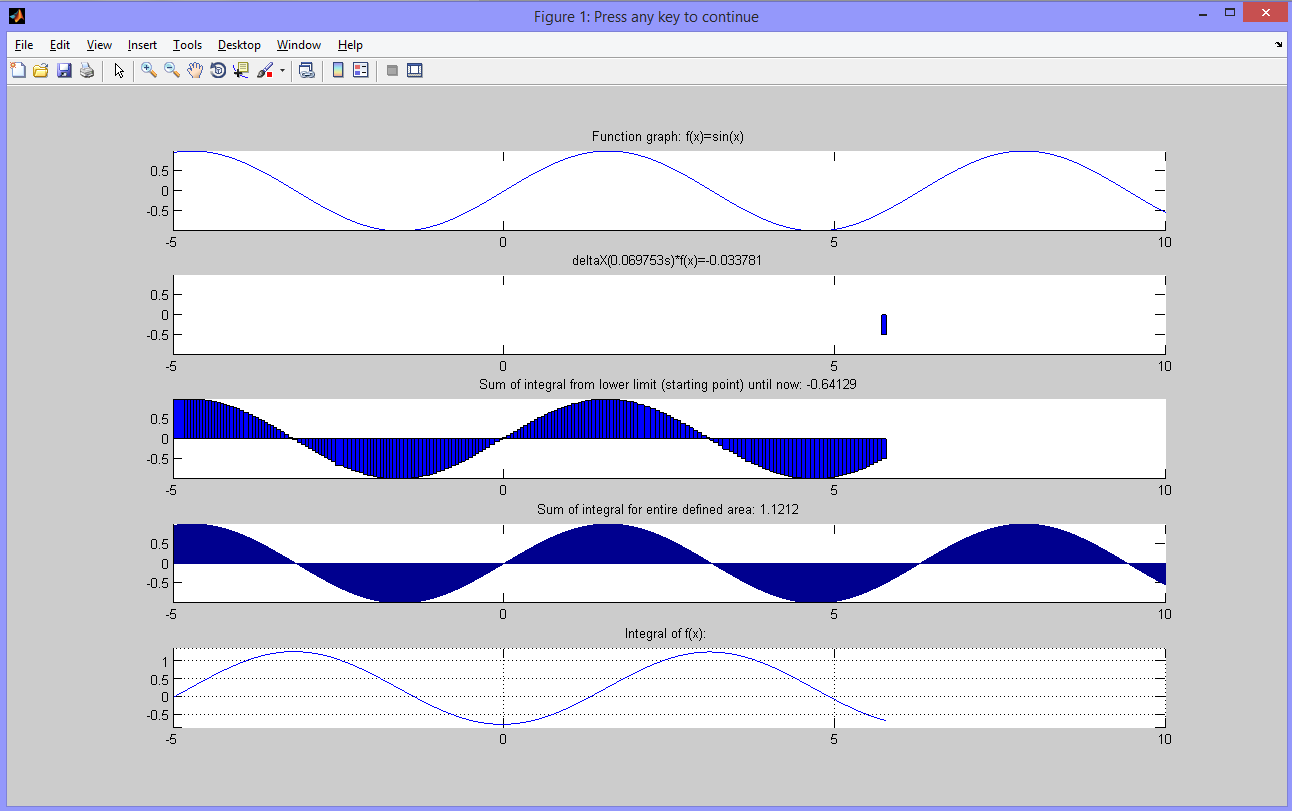
mathDerShow-funksjonen har nesten identisk logikk og kode som integral funksjonen. Plottene som vises her er. 1: selve funksjonsgrafen, 2: grafen for den deriverte fra nedre grense til nåværende løkke gjennomkjøring, 3: nåværende løkkes linæer approksimasjon og 4: alle linæer approksimasjoner.

For uten de matematiske forskjellene på de forskjellige plottene har funksjonene en forskjell i logikk. I stedenfor å benytte seg av en maks tid variabel så gjøres regnes tiden rett inn i x variablene og sammenligningen blir gjort direkte på den.

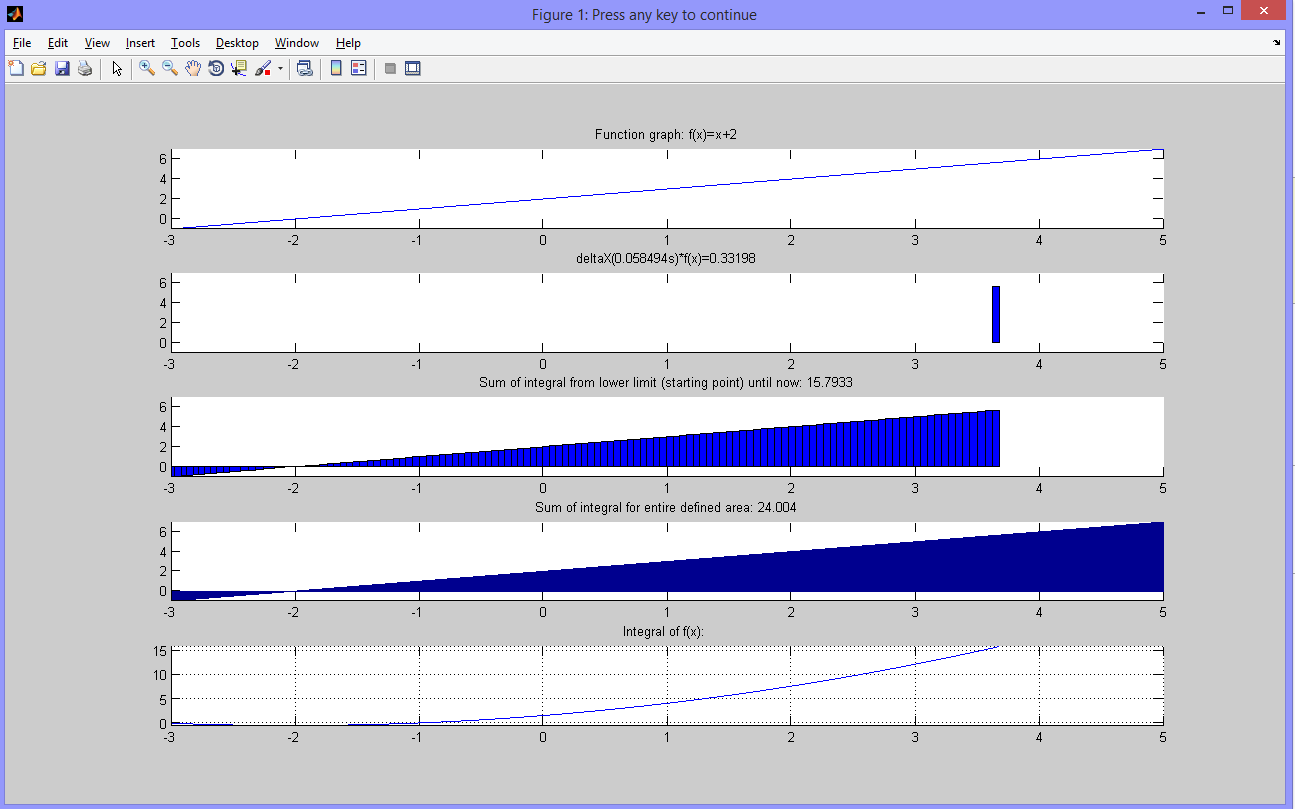
**Konklusjon mathShow:**

Programmet gir en god visualisering av noe av hva derivasjon og integrasjon faktisk er. Det gir også en klar illustrasjon av at løkketid har betydning, spesielt kommer dette godt frem for integralet hvor en kan se at rektanglene for hver løkke gjennomkjørings integral ikke er like store.

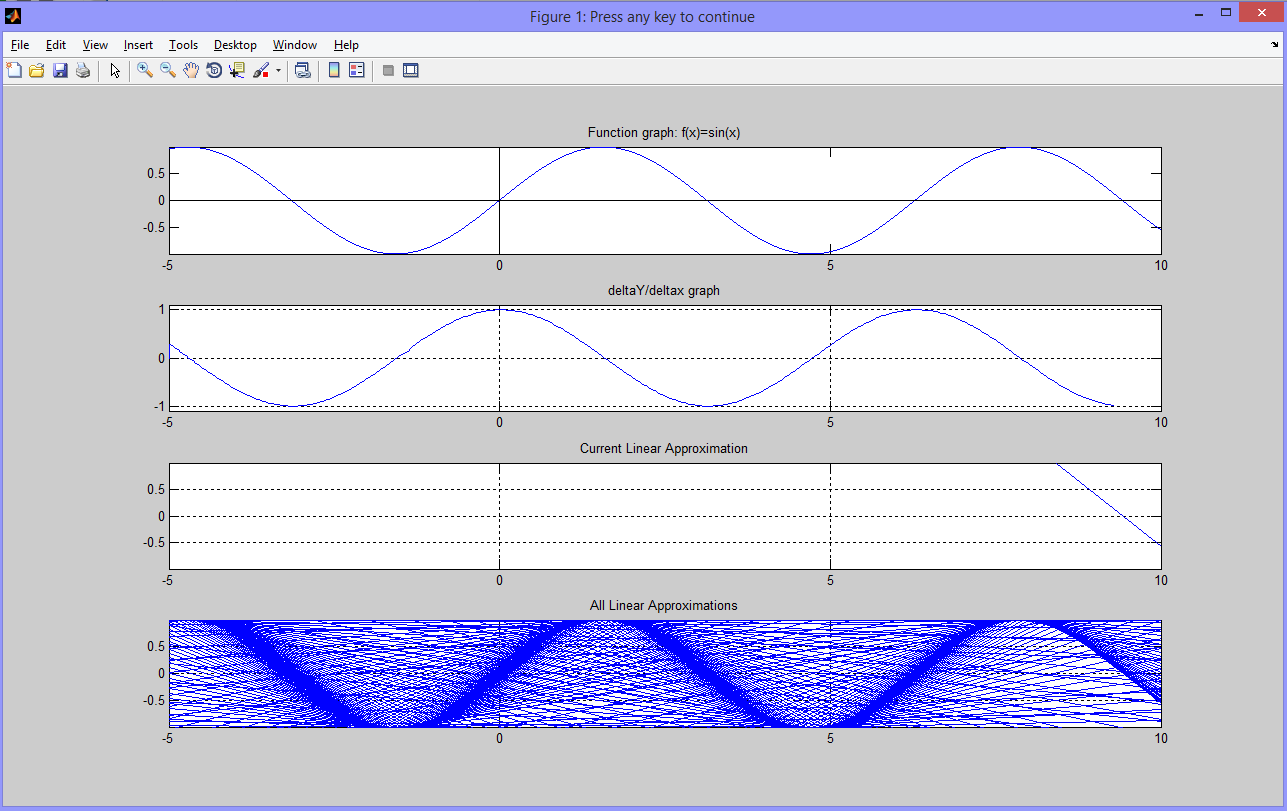
Nummerisk integrasjon av sin(x):



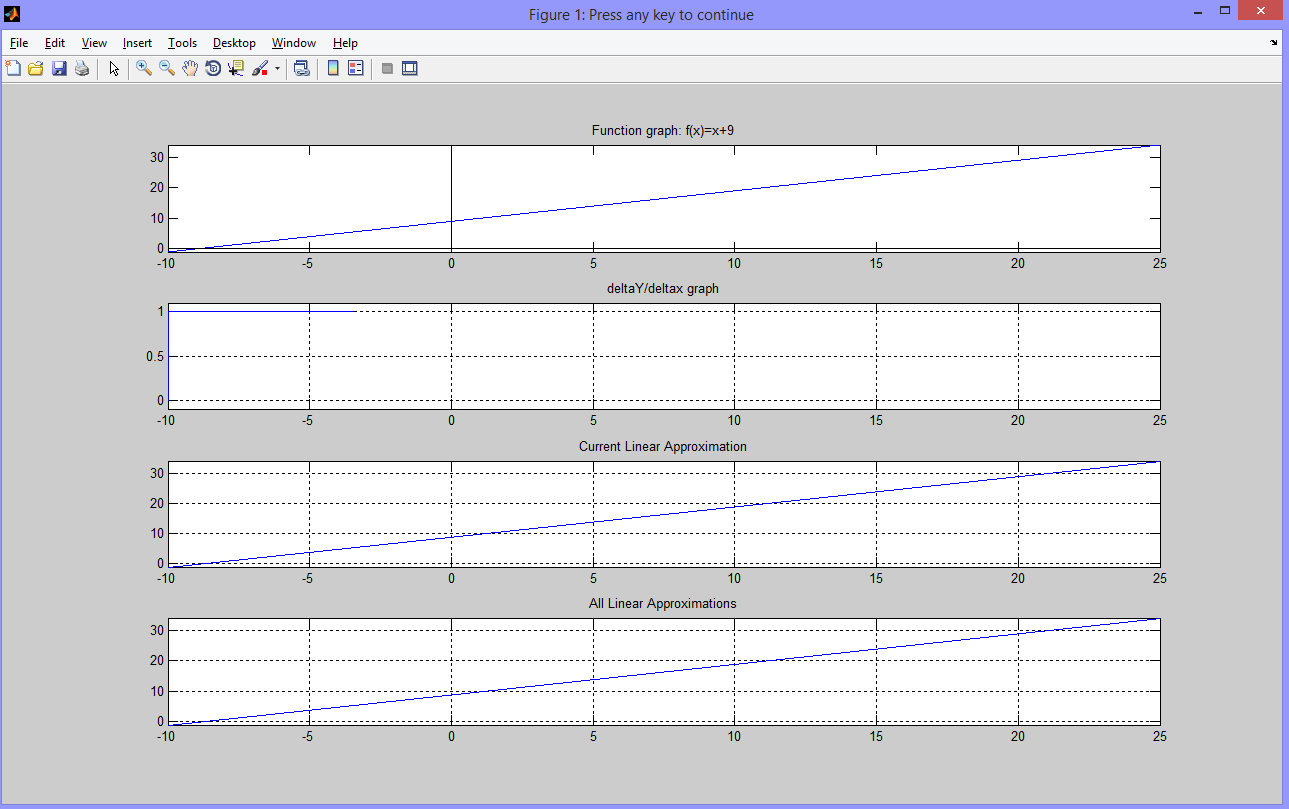
Nummerisk integrasjon av x+2:



Nummerisk derivasjon av sin(x):



Nummerisk derivasjon av x+9:



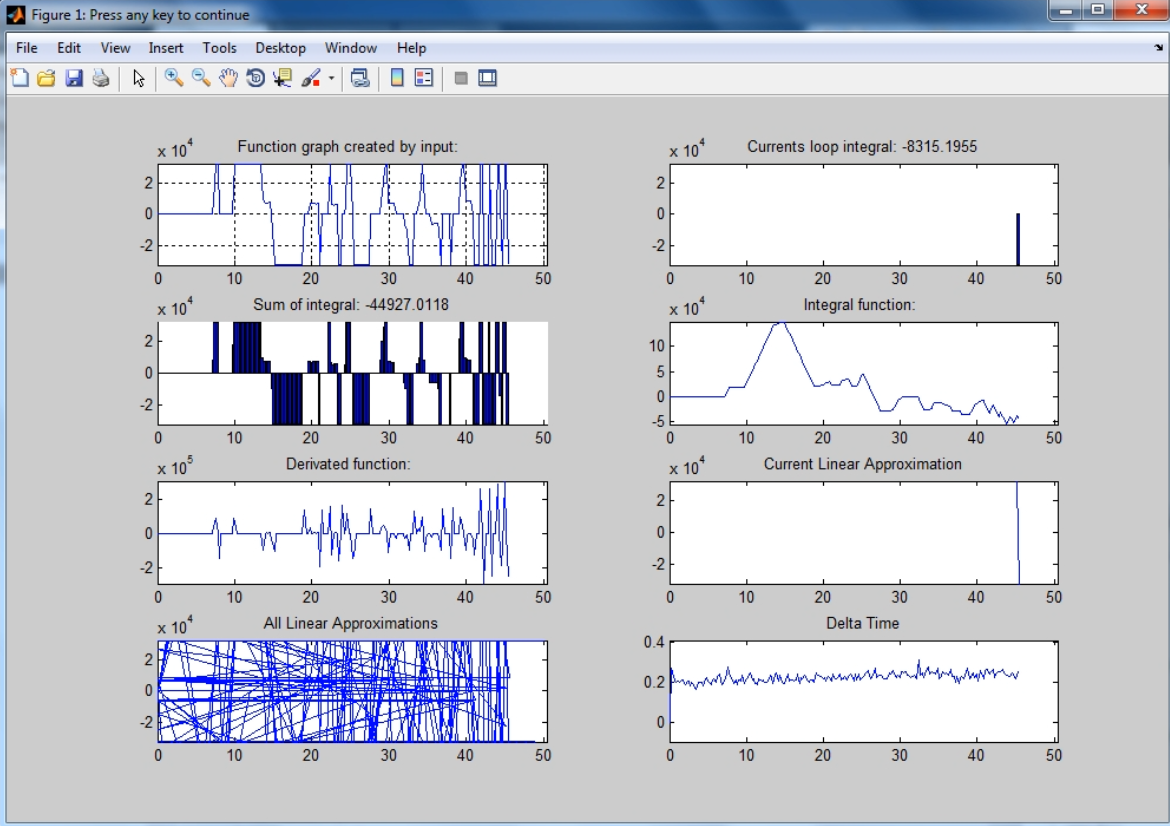
# 6. Program numJoy: (Espen Ro Eliassen)

Dette programmet bruker de samme matematiske prinsippene og logikken som i mathShow. Programmet har 8 plott som er. 1: Funksjons grafen, 2: nåværende løkke gjennomkjørings integral, 3: summen av integralene til nå, 4: funksjonen til integralet, 5: den deriverte funksjonen, 6: den nåævende linæer approksimasjonen, 7: alle linæer approksimasjoner og 8: forandringen i tid fra forige løkkegjennomkjøring til denne.

Koden i seg selv er noe mer strukturert enn i mathshow og variabler har blitt navngitt på en måte at en skal kunne benytte andre eksterne inndata og det fortsatt gir mening. Innhentingen av den eksterne dataen er også flyttet ut til en egen funksjon getJoy. Det er kun et sted denne brukes og den kan enkelt sikftes til en annen funksjon om en ønsker annen inndata. All utregning av vektor verdier skjer i starten av hver løkke gjennomkjøring og så oppdateres alle plott.

**Konklusjon numJoy:**

Programmet viser at logikken benyttet i simuleringen mathShow også virker på reelle eksterne inndata, her simulert av en joystick.



# 9. Konklusjon

Skriv noen setninger som oppsummerer programmene og hvordan de virker. Denne konklusjonen kan også deles opp etter program slik som sammendraget.

# Referanser

[1] [www.mathworks.se](http://www.mathworks.se/)

# A. Matlabfiler

Alle Matlabfiler i skriftstørrelse 6.

## A.1 Program 1 Obligatorisk del

## A.2 Program plotTime A.3 Program cannonGame A.4 Program mathShow