GUIer Gør Godt

Introduktion til GUIer i Matlab

IMM, DTU

Niels Gjøl Jacobsen

Forord

GUI er en forkortelse for 'Graphical User Interface', og det er en særdeles nyttig måde at visualisere sine data, og det gør det ligeledes på en simpel måde muligt at ændre enkelte parametrer i beregninger og se de fysiske, matematiske eller andre ændringer, som sker i forbindelse med ændringen af disse parametrer.

Ydermere hjælper GUIer til, at der bliver skabt en grænse for, hvor meget brugeren kan ændre i programmet, da der ikke direkte kontakt til den bagvedliggende kode.

Denne introduktion er skrevet til MATLAB 6.5. R13.

DTU 2003

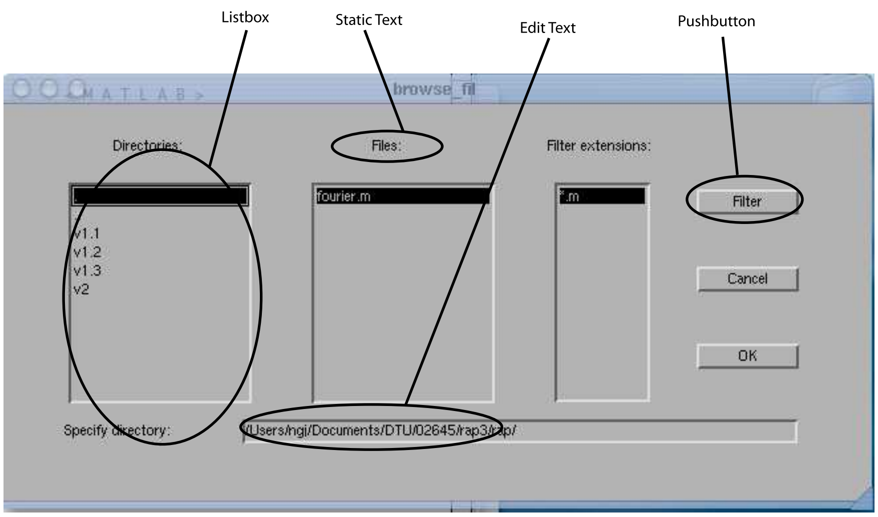
Indhold

Kapitel 1

Et lille eksempel

For at gennemgå nogle af de overordnede begreber vil et eksempel på en GUI, som alle har stiftet bekendskab med, blive gennemgået her. Den GUI, det drejer sig om, er en browse-funktion, som er særdeles nyttig, når filer skal vælges i forbindelse med andre GUIer. Selve GUIen består af to filer, som henholdsvis har `.fig` og `.m` som extension. Den første af disse to filer dannes ved at benytte et program under MATLAB, hvor det er muligt at placere de enkelte elementer i et figurfelt. Som det fremgår af figur 1.1 er der blevet brugt 4 forskellige elementer ud af 11 mulige til at lave denne browsefunktion. Alle elementerne vil blive gennemgået samlet under kapitel 2. De benyttede elementer er Listbox, Static Text, Edit Text og Pushbutton.

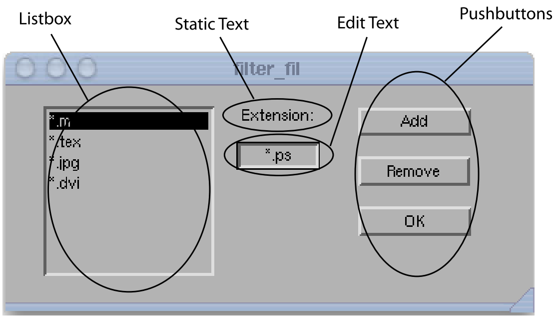
I forbindelse med dannelsen af `.fig`-filen er det ligeledes muligt at definere de forskellige elementers udseende hvad angår farver, størrelse etc.



Figur 1.1: *Udseende af browse-funktionen.*

Tilbage er der `.m`-filen, som delvist bliver genereret af MATLAB, når `.fig`-filen gemmes, men det eneste den indeholder er skabeloner til de funktioner, der herefter vil blive kaldt callbacks, som er knyttet til de enkelte elementer. Under hver af disse callbacks er det herefter muligt at programmere, hvad det enkelte element skal kunne. I forbindelse med programmeringen af disse er det min erfaring, at det er en fordel at benytte den såkaldte handles-struktur, se kapitel 3, som gør det let at føre data rundt mellem de enkelte callbacks / elementer.

Det mest interessante i denne forbindelse er dog nok muligheden for at kalde andre GUIer fra en GUI. Dette sker i dette eksempel, når der trykkes på den Pushbutton, som er behæftet med strengen *Filter*, som gør det muligt at filtrere på specifikke extensions i browsefunktionen. Når denne knap aktiveres åbnes et vindue, som kan ses i figur 1.2. Som det fremgår er det de samme 4 elementer, der



Figur 1.2: *Filterfunktionens opbygning med de samme elementer som i browsefunktionen.*

er benyttet i forbindelse med filterfunktionen. Selve det at kalde en anden GUI fra en GUI er simpelt, da `.m`-filen er en funktion med underliggende callbacks, hvorfor det bare er at kalde denne funktion fra en anden GUI. Dataoverførsel mellem de to GUI sker som sædvanligt med dataoverførsel mellem to funktioner i MATLAB.

Det ovenstående vil blive beskrevet i detaljer i de følgende afsnit.

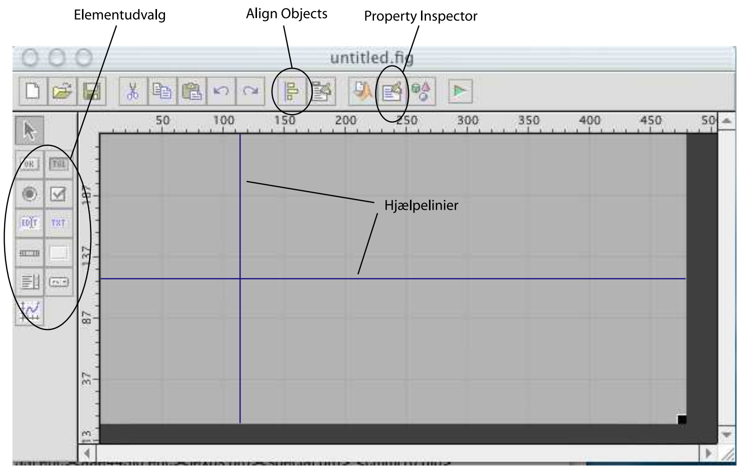
Kapitel 2

GUIDE

Til opbygning af GUIer i MATLAB er der to muligheder for at opbygge den grafiske brugergrænseflade.

Der er muligheden for selv at skrive en `.m`-fil, som angiver placering, farve og egenskaber ved de enkelte elementer. Denne fremgangsmåde vil ikke blive beskrevet i det efterfølgende. I stedet vil der blive beskrevet, hvordan den grafiske brugergrænseflade kan blive opbygget ved at benytte GUIDE, hvor det er grafisk muligt at angive lay-outet af GUIen og simpelt justere på de enkelte parametre.

GUIDE kaldes ved at skrive `guide` i kommandolinien i MATLAB, hvorved der åbner et figurfelt magen til det, som kan ses i figur 2.1.



Figur 2.1: *Sådan ser GUIDE ud når det åbnes. De vigtigste knapper samt hjælpelinier er fremhævet.*

2.1 Elementerne i GUIDE

Det er muligt at benytte 11 grafiske elementer i opbygningen af en GUI, hvor der blev stiftet bekendtskab med de 4 af dem i kapitel 1. Disse 11 elementer vil kort blive beskrevet herunder, hvorefter kodningen af nogle af dem vil blive berørt i kapitel 4, når den overordnede håndtering af handles-strukturen er beskrevet.

2.1.1 Push Button

Push Buttons bruges, når en handling skal bekræftes eller udføres, hvilket for eksempel kunne være kaldet til GUIen filter_fil under kapitel 1.

2.1.2 Toggle Button

Toggle Buttons kan benyttes, såfremt det ønskes at angive om en egenskab er aktiv eller inaktiv. Dette kan for eksempel være en zoom-knap knyttet til en Axes, i hvilken det er muligt at inkludere plots i en GUI.

For let at anskueliggøre egenskaben ved elementet kan der påsættes et icon symboliserende elementets egenskab. Dette kan for eksempel være en lup. Den bagvedliggende kode, som danner en lup med farvet baggrund, kan ses i appendix 6.3, og metoden til at indsætte disse på elementet kan ses beskrevet under afsnit 5.3.

2.1.3 Radio Buttons

Såfremt der er flere options at vælge imellem er Radio Buttons en oplagt mulighed. De enkelte Radio Buttons kan forbindes, således at alle på nær én nulstilles, når en anden vælges til værende aktiv.

2.1.4 Checkbox

Hvis flere uafhængige options skal angives som værende aktive eller inaktive i forbindelse med kørsel af en GUI er det muligt at benytte Checkboxes til dette formål.

2.1.5 Edit Text

Til at inkludere helt valgfrie parametre eller udtryk i GUIen benyttes Edit Text elementet.

Et eksempel kan være, at i forbindelse med løsningen af en differentialligning vil det være nødvendigt at have den nødvendige begyndelsesværdi som inputstørrelse.

2.1.6 Static Text

Dette element er et ikke-aktivt element, som kan benyttes til at indlægge forklarende tekst i GUIen. En anden brug kunne være at ændre den tilknyttede tekststreng løbende alt efter hvilke parametre der er valgt.

Med et ikke-aktivt elementet menes, at der ikke genereres et callback til elementet i den `.m`-fil, som Matlab genererer.

2.1.7 Slider

Ønskes der valgt en numerisk størrelse i en GUI, som ligger i et nærmere angivet interval, er det hensigtmæssigt at benytte en Slider i kombination med et Edit Text element. På denne måde er det visuelt let at se for brugeren, hvad grænserne for intervallet er. Samspillet med Edit Text elementet gør det let at indtaste specifikke numeriske størrelser, som det ikke er muligt at ramme præcist ved at bevæge Slideren. Men ligeledes giver Edit Text elementet mulighed for at vise, hvilken aktuel værdi Slideren har, når denne bevæges.

En implementering af denne egenskab er angivet i afsnit 4.5.

2.1.8 Frame

En Frame benyttes til at samle enkelte elementer og angive, at de hænger sammen i en større enhed. For eksempel en række Radio Buttons imellem hvilke der skal / kan vælges ét enkelt element.

2.1.9 Popup Menu

Hvis brugeren skal vælge imellem nogle forskellige indstillingsmuligheder er Popup Menuen et glimrende virkemiddel. Dette kunne være om en plottet kurve skal være rød, blå, gul eller grøn, hvorfor disse muligheder vil være givet som prædefinerede muligheder i Popup Menuen.

2.1.10 Listbox

Et Listbox-element er yderst anvendeligt, når der skal indhentes lange lister, eller hvis brugeren skal vælge mellem en lang række prædefinerede indstillinger. Det skyldes, at det ikke vil være hensigtsmæssigt, at benytte en Popup Menu til for mange valgmuligheder.

2.1.11 Axes

Hvis det ønskes at plotte i GUIen, skal der indsættes en eller flere Axes, i hvilke det er muligt at plotte beregnede data. Med Axes er det ligeledes ad omveje muligt at printe til fil eller direkte til printer, men hvordan de korrekte Axes bliver benyttet til plots og hvordan printet fortages, beskrives nærmere under kapitel 5.

Axes er som standard ikke et aktivt element, men udelukkende et element, som kan blive mål for en handling udført ved aktivering af et af de andre elementer.

2.2 Property Inspector

I GUIDE er det muligt at åbne et vindue kaldet Property Inspector, i hvilket det er muligt at definere alle de variable størrelser, som er gældende for de enkelte elementer. Dette værende farve, størrelse, position, tilknyttet tekststreng, etc.

På figur 2.1 er det muligt at se, hvordan Property Inspector åbnes

Den vigtigste variabel, som bør ændres inden `.m`-filen dannes første gang er elementets `tag`. Dette skyldes, at alle elementer får tildelt et `tag` med standartnavne, som kan være svære at huske. Dette kan eksempelvis være `edit1, edit2, ..., edit20`, hvis der er 20 Edit Text elementer i brug. Derfor er det vigtigt at ændre disse navne til noget, der er lettere at huske og giver større mening i forbindelse med det elementet skal bruges til i GUIen.

Et eksempel på et lettere navn kunne for eksempel være at ændre en Pushbutton's `tag` fra `pushbutton3` til `close`, såfremt elementets funktion er at lukke GUIen. Dette gør det væsentligt lettere at huske de enkelte `tags`, som skal benyttes i programmeringsfasen.

Det er meget vigtigt at bemærke, at det er besværligt at ændre et elements `tag` efter dets callback er genereret, og endvidere skal koden tjekkes, for at der ikke er referencer til netop dette callback, hvor det gamle `tag` benyttes. Så det anbefales at holde fast i de enkelte `tags`, når disse først er defineret og taget i brug.

2.3 Layout af GUI

Der er to elementer, der er vigtige at tage højde for, når lay-outet af GUIen skal færdiggøres. For det første skal opsætningen være letforståelig, så det er simpelt at gå til. For det andet skal den grafiske side af lay-outet være indbydende, hvorfor en vis konsekvens i størrelserne af elementer, farve og placering af disse er meget vigtig.

Det er udelukkende den sidste del af lay-outet, der vil blive beskrevet herunder.

2.3.1 Størrelse, farve og andet

Størrelse, farve og andet ændres lige som alle andre parametre ved brug på Property Inspector.

2.3.2 Justering af elementer

Justering af elementer kan foretages på fire forskellige måder.

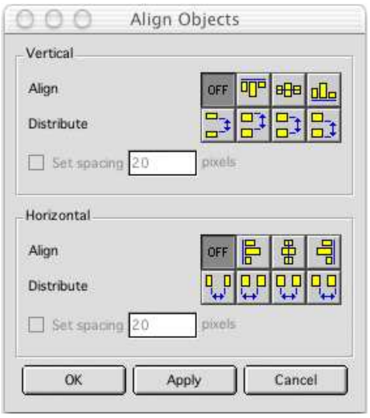
1. Under menuen `Tools` i GUIDE vælges `Grid and Rulers`. Her er det muligt at slå `Show rulers` til. Dette give en lodret og vandret lineal i figurfeltet. Ud fra disse linealer kan der trækkes hjælpelinier, hvilket giver et brugerdefineret net, hvorefter det er muligt at placere de enkelte elementer hurtigt og nemt, da elementerne 'snapper' til disse hjælpelinier.

2. Når GUIDE åbnes, er der et prædefineret net af hjælpelinier, som ligger fast, som elementerne ligeledes 'snapper' til.

3. Under menuen `Tools` er det ligeledes muligt at vælge `Align Objects`, hvorved elementer kan placeres i forhold til hinanden, på den måde det

er angivet i det figurfelt, der kommer op, når denne menu kaldes. Menuen ser ud som på figur 2.2.

4. Endelig er det muligt at placere de enkelte elementer ved at indtaste de ønskede koordinatsæt i Property Inspector.



Figur 2.2: *Sådan ser vinduet* `Align Objects` *ud.*

2.3.3 Tabulator

Da en del af layoutet også omfatter en brugervenlig rækkefølge af objekterne, når `TAB` benyttes til at springe imellem disse, er der under `Tools` en menu i hvilken det er muligt at ændre på tabulatorrækkefølgen. Denne menu hedder `Tab Order Editor`.

Som udgangspunkt er rækkefølge den, i hvilken elementerne er blevet indsat i GUIDE, og denne er ikke altid den mest hensigtsmæssige, da enkelte elementer kan have været glemt i begyndelsen.

2.4 Dannelse af `.m`-fil til programmering af GUI

Når alle de nødvendige `tags` er defineret er det på tide at danne den `.m`-fil, som skal benyttes til at definere de enkelte elementers egenskaber. Denne dannes simplest ved at vælge at gemme figuren dannet i GUIDE på normal vis ved `Save As...`.

Tilføjes der senere nye elementer til figuren i GUIDE er det intet problem. Figuren gemmes blot og et callback til elementet genereres af MATLAB og tilføjes i bunden af `.m`-filen.

Kapitel 3

Variabelstrukturen

I forbindelse med datastrømmen i en GUI er det muligt at benytte sig af tre former for variable. Det er lokale og globale variable og endelig variable, som bliver styret af den såkaldte handles-struktur. De tre typer variable vil blive behandlet herunder.

3.1 Lokale variable

De lokale variable eksisterer kun i den funktion, de er tilknyttet. Det betyder, at der er flere callbacks, som godt kan have en variabel med samme variabelnavn.

3.2 Globale variable

Globale variable har den ulempe, at de er tilgængelige overalt i MATLAB. Det vil sige, at kørsel af andre funktioner i MATLAB, der kører sideløbende med GUIen, også har adgang til disse variable. Dette kan være uønsket, da der så ikke er 100% styr på variablerne. Globale variable kan ikke anbefales, da handles-strukturen giver større muligheder. Der kan fås mere information ved at skrive `help global`.

3.3 Handles-strukturen

Alle elementer har en `handle`, og det er dette GUIen bygger på. Denne `handle` indeholder alle de data, som bliver defineret i forbindelse med Property Inspector, se afsnit 2.2, og den enkelte `handle` ligger gemt i et strukturfelt, som indeholder `handles` på samtlige elementer.

Alle variable i dette strukturfelt har det samme fornavn, nemlig `handles.*`, hvor stjernen ikke skal opfattes som om, at alle filformater er tilladte, men at alle efterfølgende lovlige variablenavne er tilladte. For eksempel vil en Push Button automatisk få tildelt `handles.pushbutton1`, når den dannes i GUIDE.

Dette betyder, at de variable, som er tilknyttet elementer i GUIen, har samme efternavn som deres `tag`.

Det kan i MatLab simpelt tjekkes om et variablenavn er tilladt ved at benytte kommadoen `isvarname('string')`.

3.3.1 Variable med handles-strukturen

Ud over at der er tilknyttet et `handle` til samtlige callbacks / elementer, kan der oprettes vilkårligt mange `handles` til at gemme data i. Fremgangsmåde vil blive vist med et eksempel.

Det tænkes, at der i en Popup Menu dannes en række data, som sidenhen skal plottes, når der trykkes på en Push Button. Dette gøres på følgende måde, såfremt data ønskes overført via handles-strukturen mellem de forskellige callbacks. Hvis dette ikke er tilfældet, se da afsnit 3.4.

```
% --- Executes on selection change in popup.
function popup_Callback(hObject, eventdata, handles)
% hObject    handle to popup (see GCBO)
% eventdata  reserved - to be defined in a future version of
                                                     MATLAB
% handles    structure with handles and user data
                                              (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popup
                                       contents as cell array
%        contents{get(hObject,'Value')} returns selected item
                                                 from popup
    x = linspace(0,1,100);
    x = x(:);
    handles.uddata = [x, sin(x)];
    guidata(hObject,handles);

% --- Executes on button press in integrate.
function pushbotton1_Callback(hObject, eventdata, handles)
% hObject    handle to integrate (see GCBO)
% eventdata  reserved - to be defined in a future version of
                                                     MATLAB
% handles    structure with handles and user data
                                              (see GUIDATA)
    figure
    plot(handles.uddata(:,1),handles.uddata(:,2));
```

Her opdaterer `guidata(hObject,handles)` alle de `handles`, som er blevet defineret i forbindelse med den igangværende callback og gemmer disse i strukturfeltet. Når de så ønskes hentet frem, ligger de umiddelbart tilgængelige for samtlige callbacks i denne GUI, hvorfor de direkte kan kaldes.

Det er umiddelbart den simpleste måde at håndtere datastrømmen mellem de enkelte callbacks / elementer.

3.4 Andre måder at dele data mellem de enkelte elementfunktioner

Der er dog andre måder at dele data imellem de enkelte elementer på. Her benyttes `handlen` til et enkelt element til at gemme data i.

Et eksempel kunne være, at den streng, som er indtastet i et Edit Text element skal benyttes til nogle udregninger. Under forudsætning af at strengen er en numerisk størrelse kan værdien direkte evalueres ved at skrive

```
value = eval(get(handles.edit1,'string'));
```

Hermed kan den uden videre bruges i beregningerne i den pågældende callback. Hvis værdien skal bruges i `function edit1_callback(..,..,..)` kan dette kan også skrives som

```
value = eval(get(hObject,'string'));
```

da et kald til `handles` i det callback tilknyttet samme element kan fortages med `hObject` i stedet for `handles.<tag>`.

At `eval` benyttes frem for `str2double` til at omforme strengen til en numerisk størrelse skyldes udelukkende, at `str2double(pi)` giver en fejlmeddelelse, men `eval(pi)` returnerer $3.1415\ldots$.

Vær dog opmærksom på, at `eval` ikke kan benyttes, såfremt strengen har indgået i et `cell array`. For at komme uden om dette problem omformes strengen ved hjælp af `char` på følgende måde

```
value = eval(char(get(handles.edit1,'string')));
```

Det er ligeledes muligt at overføre en større mængde data via den eksistrende handles-struktur. Dette kan gøres ved at benytte UserData, som er en del af alle elementer i GUIen. At gemme data i denne størrelse kan gøres ved at skrive

```
set(hObject,'UserData',value)
```

hvis størrelsen `value` skal gemmes i den igangværende callback's `handle`.

Til dette kan det anbefales at benytte cell array, hvis der er flere data, som skal overføres. For mere information brug `help` på `cell`, `cell2mat` og `deal`. Dog vil et lille eksempel være illustrativt.

Det antages, at to matricer defineret som

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 7 & 4 & 5 \\ 3 & 9 & 11 & 1 \\ 4 & 1 & 20 & 2 \end{bmatrix}$$

ønskes overført via UserData. Da A og B har vidt forskellige dimensioner, kan de ikke umiddelbart sammenlægges i én variabel og overføres gennem UserData for derefter at splittes op. Her viser `cell arrays` deres brugbarhed. Skrives for eksempel

```
g = {A,B};
set(hObject,'UserData',g);
```

dannes en variable indeholdende de to variable A og B. Når overførslen er sket skal der blot skrives

```
g = get(handles.pushbutton1,'UserData');
A = cell2mat(g(1)); B = cell2mat(g(2));
```

og de to variable A og B er belvet overført fra `handles.pushbutton1` til den pågældende callback, hvor de skal benyttes.

En GUI lukkes, ved at der skrives `delete(handles.figure1)` et sted i `.m`-filen, hvor `figure1` er det `tag`, som er tilknyttet selve GUIen, hvorved der menes baggrunden af GUIen.

Nu skulle de mest overordnede elementer være på plads. I kapitel 4 bliver det beskrevet, hvordan nogle af elementerne kan programmeres, og i kapitel 5 gennemgås eksempler, som kan gøre GUIen mere brugervenlig, såsom brug af dialogbokse og diverse advarsler.

Kapitel 4

Programmering af elementer

I dette kapitel findes beskrivelse af måder at programmere de enkelte elementer på. Da Static Text, Frames og Axes ikke direkte bliver tilknyttet et callback vil programmering af disse ikke blive taget med her. Det er dog stadig muligt at ændre i disse elementers egenskaber ved at benytte `set(handles.<tag >,option,value)` fra et andet callback. Eksempelvis vil indholdet i et Static Text element kunne ændres løbende.

4.1 Push Button

Der er intet at bemærke til programmeringen af en Push Button, da denne mest benyttes til at samle trådene efter diverse indstillinger. Derfor skal de kommandoer, der ønskes aktiveret, eller de funktioner, der ønskes kaldet, blot implementeres på sædvanlig måde.

4.2 Toggle Button

En Toggle Button kan eventuelt bruges, hvis det ønskes at slå zoom tilknyttet en specifik Axes til og fra. Dette gøres ved

```
function toggle1_Callback(hObject, eventdata, handles)

axes(handles.axes1);          % aktivere axes1
val = get(hObject, 'value'); % henter Toggle Button værdien

if val == 1
    zoom on;
else
   zoom off;
end              % Slår zoom til og fra
```

4.3 Radio Button

Ved benyttelse af Radio Buttons skal alle andre sammenhængende Radio Buttons nulstilles på nær en. Her er det fornuftigt at benytte nummerede `tags`, således at nulstillingen kan foretages i en løkke. Dette gøres ved

```
function radiobutton1_Callback(hObject, eventdata, handles)

val = get(hObject,'value');
if val == 1
    for i={'2','3','4'}
        eval(strcat('set(handles.radiobutton',char(i),
                                                    ',''value'',0)'));
    end
end
```

En anden mulighed ville være at definere følgende `handle`, når der arbejdes på en GUI, som kaldes `forsoeg.m`. Til enhver GUI vil der i `.m`-filen være et åbnings-callback, som i dette tilfælde hedder

```
function forsoeg_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;
handles.group_radio = [handles.radiobutton1; ...
                       handles.radiobutton2; ...
                       handles.radiobutton3];
guidata(hObject, handles);
```

Her er det eneste, der er tilføjet, definitionen på det nye `handle`. Det er herefter muligt, analogt til den anden fremgangsmåde, at nulstille værdien for 2 ud af de 3 Radiobuttons på følgende måde

```
function radiobutton1_Callback(hObject, eventdata, handles)

val = get(hObject,'value');
if val == 1
    set(handles.group_radio(1),'value',1);
    set(handles.group_radio(2:3),'value',0);
end
```

4.4 Checkbox

Forestiller man sig, at der skal benyttes en Checkbox til at angive om zoom er slået til eller ej vil programmeringen være meget analog til den for Togglebuttons. Denne vil være givet som

```
function checkbox1_Callback(hObject, eventdata, handles)

axes(handles.axes1);          % aktivere axes1
val = get(hObject, 'value'); % henter Checkbox værdien
if val == 1
    zoom on;
```

```
else
   zoom off;
end            % Slår zoom til og fra
```

4.5 Interaktion mellem Edit Text og Slider

I forbindelse med programmering af GUIer er det ikke muligt at udgå interaktion mellem de enkelte elementer. Det følgende eksempel går på, at der er en Slider, som kan antage værdier i et ikke nærmere defineret interval. Desuden er der et Edit Text element, i hvilket der kan indtastes en værdi, som ligger inden for intervallet, hvorefter Slideren bliver opdateret til denne værdi. Omvendt vil Edit Text elementet bliver opdateret, såfremt der bliver ændret på Slideren.

Figur 4.1: *Eksempel på interaktion mellem Slider og Edit Text element.*

Koden til de to funktioner for henholdsvis Slider og Edit Text ser ud på følgende måde.

```
function slider1_Callback(hObject, eventdata, handles)

set(handles.edit1, 'string',num2str(get(hObject,'value')))

%%%%%%%%

function edit1_Callback(hObject, eventdata, handles)

val = eval(get(hObject,'string'));
int = [get(handles.slider1,'min'),get(handles.slider1,'max')];

if val <= int(2) & int(1) <= val
    set(handles.slider1,'value', val);
else
    uiwait(warndlg('The printed value is not in the demanded
                         interval','Warning','modal'))
    uiresume
end
```

Hvor den sidste `else`-sætning printer en fejlmeddelelse såfremt den indtastede værdi ikke ligger i intervallet `[int(1),int(2)]`. Hvordan sådanne benyttes og hvilke andre der findes kan ses under afsnit 5.2.

4.6 Listbox

I forbindelse med programmering af Listbox er det mest interessante at se på hvordan en sådan opdateres, som det eksempelvis skete i eksemplet i kapitel 1. Det er ikke interessant at se på, hvordan algoritmen ser ud for prædefinerede størrelser, da fremgangsmåde er identisk med den for Popup Menuen.

Den tekst, der står i Listboxen, kommer fra et `cell array` af dimensionen $n \times 1$, hvilket giver n linier i Listboxen. Når Listboxen så opdateres, så arrayet er $k \times 1$ og $k < n$, kan der opstå problemer. Det skyldes, at såfremt det linienummer, der er highlighted gemmes, og hvis linienummeret er større end k kan den samme linie ikke være aktiv efter opdateringen, og GUIen afgiver en fejlmeddelelse. Derfor anbefales det altid at angive linienummeret som værende 1 umiddelbart før opdateringen. Den option det drejer sig om er `'value'`. Et eksempel kunne se ud som følger, hvor `streng` er den streng, der opdateres med.

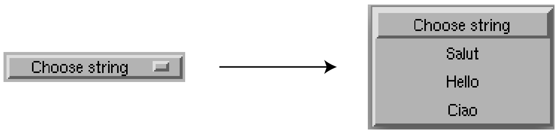
```
function listbox1_Callback(hObject, eventdata, handles)

streng = {'hej','hello','ciao','guten tag'};
set(hObject,'value',1);
set(hObject,'string',streng);
```

Samme problematik skal der være opmærksomhed omkring ved opdatering af Popup Menu.

4.7 Popup Menu

Indholdet i en Popup Menu er oftest defineret på forhånd i forbindelse med kodningen, så det er kun interessant at beskrive, hvorledes kodningen ser ud for at den algoritme svarende til den valgte størrelse kan udføres. Se eksempel vis på en Popup Menu, som den på figur 4.2. Hvis den enkelte hilsen ønskes



Figur 4.2: *Popupmenu. Henholdsvis ikke aktiv og aktiveret.*

udskrevet på skærmen, og at der skal komme en fejlmeddelelse, når muligheden 'Choose string' vælges, ser kodningen ud som følger.

```
val = get(hObject,'value');

string = get(hObject,'string');
switch val
    case 1
        uiwait(warndlg('Please choose a word below.'
                                                ,'OBS!','modal'))
```

```
        uiresume
    case {2,3,4}
        disp(char(string(val)));
end
```

Der er ikke den store forskel mellem denne kode, og hvis der skal udføres større beregninger ved det valgte. Den eneste forskel er hvad der står under den enkelte `case`.

Kapitel 5

Lidt ekstra

5.1 Dataoverførsel mellem GUIer

Dataoverførsel mellem to GUIer foregår ved at kalde en `.m`-fil fra den aktive GUI. Denne `.m`-fil er logisk nok koden hørende til en anden GUI. Ligesom med dataoverførsel via `UserData` kan en GUI kun modtage en variabel som input, hvorfor samme metode benyttes, se afsnit 3.4. Dog er det muligt at sende vilkårligt meget data tilbage, hvilket letter opgaven en smule.

Den GUI, hvor kaldet foretages, kaldes i det efterfølgende GUI-1, og den der kaldes får navnet GUI-2. Kode i GUI-1, når der kaldes en funktion kaldet `input_file.m`:

```
S = {a,b,c};
[A,B,C] = input_fil(S);
```

I GUI-2 sker da følgende, hvor dens åbnings-callback ser ud på følgende måde:

```
function input_file_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for input_file
handles.output = hObject;
handles.a = (varargin{1}{1});
handles.b = (varargin{1}{2});
handles.c = (varargin{1}{3});
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes input_file wait for user response (see UIRESUME)
% uiwait(handles.figure1);
uiwait(handles.figure1);
```

Det væsentlige her er omformningen af data, og endvidere det vigtige, at kommandoen `uiwait(handles.figure1)` benyttes. Dette betyder, at GUIen ikke som default returnerer en ligegyldig værdi, men venter til at de variable, der skal returneres, er genereret. Dette kunne for eksempel ske som følger

```
function varargout = input_file_OutputFcn(hObject, eventdata,
                                                  handles)
```

```
% Get default command line output from handles structure
varargout{1} = handles.a;
varargout{2} = handles.b;
varargout{3} = handles.c;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

handles.a = handles.a * 2;
handles.b = handles.b * pi;
handles.c = handles.c * 3;
guidata(hObject,handles);
uiresume(handles.figure1)
```

Her finder det mest interessante sted med kommandoen `uiresume(handles.figure1)`, for det giver GUIen lov til at eksekvere det callback, der hedder `input_file_OutputFcn`, som returnerer de variable, der skal returneres.

Dette er helt overordnet den fremgangsmåde, der skal benyttes med dataoverførsel mellem GUIer.

5.2 Dialogbokse

Dialogbokse kan være særdeles anvendelige, hvis brugeren skal gøres opmærksom på en opstået fejl, eller der er brug for, at brugeren verificerer sit valg. Eksempelvis kan der ønskes bekræftet, om GUIen skal lukkes eller ej. Dialogbokse kan være de i MatLab indbyggede, ellers kan de konstrueres efter ens egne ønsker.

5.2.1 Indbyggende dialogbokse

De dialogbokse som vil blive behandlet her, er dem, som er indbygget i Matlab. De fungerer generelt ens, hvorfor der henvises til `help` på `msgbox`, `questdlg`, `errordlg`, `warndlg`, `helpdlg` og `textwrap`.

Herunder vil der dog blive givet et eksempel på, hvorledes sådan en dialogboks vil kunne se ud. Det vil dreje sig om en forespørgsel til brugeren om det ønskes, at programmet skal lukkes, hvor valgmulighederne er 'Yes', 'No' eller 'Cancel'. Dette gøres på følgende måde

```
function pushbutton1_callback(hObject, eventdata, handles)
% denne funktion lukker GUIen

    quest = questdlg({'Do you really want','to close this current',
                                'session?',''},'Question','No');
    if quest == 'Yes'
       delete(handles.figure1)
    end
```

Kaldet til `questdlg` åbner en dialogboks, se figur 5.1, i hvilken der er en 4-liniers tekst, hvoraf der er tekst på 3 af dem. Det er fordelen ved at benytte `cell array` til at indtaste en tekststreng, da det kan styres, hvor strengen skal deles. Den sidste del af kaldet, hvor der står skrevet `'No'` angiver hvilken af de tre valgmuligheder, der skal være prædefineret som default.

Figur 5.1: *Den beskrevne dialogboks.*

5.3 Indsættelse af iconer

Iconer indsættes på de enkelte elementer på en meget simpel måde. Dette gøres ved at danne en M×N×3 matrix i MATLAB udelukkende indeholdende værdier i intervallet $[0, 1]$, og gemme denne matrix som en variabel. Denne matrix definerer billedet i farver. Intervallet er netop $[0, 1]$, da en farve i MATLAB er defineret som RGB farve, hvor hver af farverne rød, grøn og blå repræsenteres af en numeriske værdi mellem netop 0 og 1.

Herefter markeres det element i GUIDE, som iconet skal tilknyttes og i Property Inspector findes den parameter, som hedder `CData`. Når denne vælges kommer der et figurefelt op, i hvilket der står `Enter expression`, hvor variabelnavnet skrives. Hermed er iconet tilføjet.

Det er dog væsentligt, at variablen ligger i workspace, da den ellers ikke vil optræde som en kendt variable, og derfor ikke vil være mulig at inkludere som et icon.

5.4 Print af plot i en GUI

Jeg vil nu komme med en meget besværlig fremgangsmåde, men det er ifølge Mathworks den eneste mulighed, da det ikke er muligt at printe en Axes, men kun et helt figurfelt, hvilket vil mene hele GUIen. Den kode, der skal bruges er

```
figure      % Create a new figure
axes        % Create an axes object in the figure
new_handle = copyobj(handles.printet,gca);
eval(char(strcat('print -depsc',{''},{' '}, <file_name>)));
close(gcf)
```

hvor `handles.printet` er et `handle`, som er genereret i forbindelse med plottet af det, som ønskes printet. Dette `handle` er genereret på følgende måde

```
handles.printet = plot(x,y);
guidata(hObject, handles);
```

Det, der reelt sker i forbindelse med afvikling af denne printkommando, er linie for linie:

1. Der åbnes et nyt figurvindue.
2. Der åbnes en Axes i dette figurvindue.
3. Det som ligeledes er plottet i GUIen kopieres over i den nye Axes.
4. Denne figur er et normalt figurfelt, så det kan plottes uden at overflødige ting kommer med.
5. Dette midlertidige figurfelt lukkes og man er tilbage i GUIen.

Besværligt ja, men det virker.

5.4.1 Flere Axes i en GUI

Når der er flere Axes i en GUI skal man sikre sig, at det er den rigtige Axes, der bliver plottet i. Dette klares simpelt ved at benytte følgende kommando før hvert plot

```
axes(handles.<tag>)
```

Dette gør, at den Axes med det pågældende `tag` aktiveres, og det er i denne plottet foretages.

Kapitel 6

Appendix

6.1 Små tricks

- I MatLabs teksteditor er det muligt at springe rundt i funktionerne ved hjælp af `Show function`, som kan findes i værktøjsbjælken.
- Hvis det ønskes at opdatere `handle` i et callback, der tilhører det element, som ønskes opdateret, behøver det ikke at blive kaldt med `handles.<tag>`, men det er tilstrækkeligt at skrive `hObject`.

6.2 Ting, som skal huskes

- Husk at ændre `tags` inden `.m`-filen dannes, da det er besværligt at ændre senerehen.
- Husk at units på alle elementer skal være identiske for at kunne bruge GUIen på computere med forskellige skærmopløsninger.

6.3 Kode til frembringelse af lup-icon

```
% function LUP
%
% Ved at kalde denne funktion faas data til en togglebuttom i
                                                     Matlab GUI,
% som passer i stoerrelsen 19 * 19 pixles. Data indlaegges i GUIen
                                                         ved at
% indlaese dataen i CData. Dette goeres ved, at indtastes
                                                  variabelnavnet
% under 'Enter expression'.
% colordef er en vektor indeholdende 3 elementer, nemlig
                                                 colordef = [red,
% blue, green], hvor disse stoerrelser hver ligger i intervallet
                                                         [0, 1].

function b = lup(colordef)
```

```
b(:,:,1) = colordef(1) * ones(14); b(:,:,2) = colordef(2) * ones(14);
b(:,:,3) = colordef(3) * ones(14);

b(2,5:7,:) = 0; b(3,4,:) = 0; b(3,8,:) = 0; b(4,3,:) = 0; b(4,6,:) = 0;
b(4,9,:) = 0; b(5,2,:) = 0; b(5,6,:) = 0; b(5,10,:) = 0; b(6,2,:) = 0;
b(6,4:8,:) = 0; b(6,10,:) = 0; b(7,2,:) = 0; b(7,6,:) = 0;
b(7,10,:) = 0; b(8,3,:) = 0; b(8,6,:) = 0; b(8,9,:) = 0; b(9,4,:) = 0;
b(9,8:10,:) = 0; b(10,5:7,:) = 0; b(10,9:11,:) = 0; b(11,10:12,:) = 0;
b(12,11:13,:) = 0; b(13,12:13,:) = 0;
```