Outline

- Announcements:
  - Homework II: Solutions on web
  - Homework III: due Wed. by 5, by e-mail
- Structs
- Differential Equations
- Numerical solution of ODE’s
- Matlab’s ODE solvers
- Optimization problems
- Matlab’s optimization routines

Applications of Cell-arrays

- Strings
  - charcell=({'Greetings'; 'People of Earth'})
  - disp(charcell) will output each cell on a new line
    - ‘Greetings’
    - ‘People of Earth’
  - Search lines using strmatch
    - strmatch('Greetings',charcell)
- Encapsulating data
  - FFTcell=(a, b, dt)
    - myfft could return a cell array in this form
    - Would keep the related data together
    - But, user would have to know what is in each cell
Structs

- Rather than storing data in elements or cells, structs store data in fields
- Better encapsulation than cell
  - FFTstrct.a=a;
  - FFTstrct.b=b;
  - FFTstrct.dt=dt;
- Can have arrays of structs
  - for j=1:n
    - strctarr(j)=StrctFunc(inputs);
  - end
  - each element must have same fields

Working with Structs

- Lots of commands for working with structs
  - fieldnames(S)--returns a cell array of strings containing names of fields in S
  - isfield(S,'fname')--checks whether fname is a field of S
  - rmfield(S,'fname')--removes fname

Differential Equations

- Ordinary differential equations (ODE’s) arise in almost every field
- ODE’s describe a function y in terms of its derivatives
- The goal is to solve for y
Example: Logistic Growth

- N(t) is the function we want (number of animals)

\[ \frac{dN}{dt} = rN\left(\frac{K - N}{K}\right) \]

Numerical Solution to ODEs

- In general, only simple (linear) ODEs can be solved analytically
- Most interesting ODEs are nonlinear, must solve numerically
- The idea is to approximate the derivatives by subtraction

Euler Method

\[ \frac{dN}{dt} = f(N, t) \]

\[ \frac{N_{t+1} - N_t}{\Delta t} = f(N_t, t) \]

\[ N_{t+1} = N_t + \Delta t \cdot f(N_t, t) \]
Euler Method

- Simplest ODE scheme, but not very good
- "1st order, explicit, multi-step solver"
- General multi-step solvers:

\[ N_{n+1} = N_n + \Delta t \cdot f \] (weighted mean of \( f \) evaluated at lots of \( t \)'s)

Runge-Kutta Methods

- Multi-step solvers--each \( N \) is computed from \( N \) at several times
  - can store previous \( N \)'s, so only one evaluation of \( f \)/iteration
- Runge-Kutta Methods: multiple evaluations of \( f \)/iteration:

\[
\begin{align*}
    a &= \Delta t f(N_n, t) \\
    b &= \Delta t f(N_n + a/2, t + \Delta t/2) \\
    c &= \Delta t f(N_n + b/2, t + \Delta t/2) \\
    d &= \Delta t f(N_n + c, t + \Delta t) \\
    N_{n+1} &= N_n + 1/6 \cdot (a + 2b + 2c + d)
\end{align*}
\]

ODE Solvers

- LMS and RK solvers are general algorithms that can work with any ODE
- Need a way to pass the function \( f \) to the solver
Passing Functions

- One sol’n: force us to call our function “f”
- A better sol’n:
  - pass the name of the function to the solver
    - funcname="myf"
    - solver uses “feval” command
      - dN=feval(funcname,N,t);
    - This is Matlab ca. 1998

Passing Functions

- Now, feval accepts “function handles”
  - @myf creates a handle to myf.m
- Slightly faster than strings
- Now, how do we solve the ODE’s?

Matlab’s ODE solvers

- Matlab has several ODE solvers:
  - ode23 and ode45 are "standard" RK solvers
  - ode15s and ode23s are specialized for "stiff" problems
  - several others, check help ode23 or book
### Matlab’s ODE solvers

- All solvers use the same syntax:
  - `[t,N]=ode23(@odefile, t, N0, (options, params ...))`
  - `odefile` is the name of a function that implements `f`
  - `f=odefile(t, N, {params});` is a column vector
  - `t` is either `start time, end time` or a vector of times where you want `N`
  - `N0= initial conditions`
  - `options` control how solver works (defaults or okay)
  - `params= parameters to be passed to odefile`

### Parameters

- To accept parameters, your ode function must be polymorphic
  - `f=odefile(t,x);`
  - `f=odefile(t,x,params);`
  - `function f=odefile(t,x,params);`
    - `if(nargin<3)`
      - `params=defaults;`
    - `end`
    - `f=...`

### nargin

- Matlab functions can be polymorphic--the same function can be called with different numbers and types of arguments
  - Example: `plot(y), plot(x,y), plot(x,y,’rp’);`
- In a function, `nargin` and `nargout` are the number of inputs provided and outputs requested by the caller.
- Even more flexibility using `varargin` and `varargout`
Example: Lorenz equations

- Simplified model of convection cells
  \[
  \begin{align*}
  \frac{dx}{dt} &= \sigma (y - x) \\
  \frac{dy}{dt} &= \tau z - y - xz \\
  \frac{dz}{dt} &= \pi y - 6z
  \end{align*}
  \]
  - In this case, \( N \) is a vector \([x, y, z]\) and \( f \) must return a vector \([x', y', z']\)

Optimization

- For a function \( f(x) \), we might want to know
  - at what value of \( x \) is \( f \) smallest
  - largest?
  - equal to some value?
- All of these are optimization problems
- Matlab has several functions to perform optimization
  - Same problem as with ODE: solver needs to work with YOUR functions
  - Must pass function handle

Optimization

- Simplest functions are
  - \( x = \text{fzero}(\text{@fun, } x0); \) \% Finds \( f(x) = 0 \) starting from \( x0 \)
  - \( x = \text{fminbnd}(\text{@fun, } [xL, xR]); \) \% Finds minimum in interval \([xL, xR]\)
  - \( x = \text{fminsearch}(\text{@fun, } x0); \) \% Finds minimum starting at \( x0 \)
- All of these functions require you to write a function implementing \( f \):
  - \( \text{function } f = \text{fun}(x); \% \text{computes } f(x) \)
Optimization Toolbox

- More optimization techniques, but used in the same way
  - Some of these functions can make use of gradient information
  - in higher dimensions, gradient tells you which way to go, can speed up procedure

![Graph showing a function with derivative f and df/dx = 0]

Optimization Toolbox

- To return gradient information, your function must be polymorphic:
  - x=fun(x) %return value
  - [x,dx]=fun(x); %return value and gradient
  - function [x,dx]=fun(x);
    - if(nargout>1)
      - dx=...
    - end
    - x=...

Optimization Options

- Need to tell optimization toolbox that it can use gradient
  - Use optimset:
    - opts=optimset('ParameterName','value');
    - Then pass opts to optimization function
  - Lots of options--read docs!
  - opts=optimset('GradObj','on'); %Turns gradient on
  - x=fminunc(@fun, x0,opts); %pass opts to function
Summary

- ODE solvers & optimization routines in Matlab are "function functions"
  - You must write a function returning
    - \( \frac{dx}{dt} \) (ODE)
    - \( f \) (Optimization)
  - You pass the name of the function to the solver
  - Solver calls it repeatedly and returns answer