G53FUZ
Fuzzy Sets and Systems

Fuzzy Modelling and Tuning

Bob John
ASAP
Research Group

Overview

- Fuzzy model identification
  - structure & parameters
  - evaluation
  - tuning example
- Tuning methods
  - exhaustive search
  - monte carlo
  - hill climbing and stochastic local search
  - simulated annealing and simplex method

Model Tuning

Fuzzy Model Identification

- Finding a good fuzzy model can be formulated in terms of searching for a candidate solution from a (very) wide range of possibilities
  - model identification
  - model optimisation
  - model tuning
- There are very many design parameters in any fuzzy model
  - perhaps this is the ‘secret’ of success?

Structure or Parameters?

- Some people divide problem into two parts
  - structure identification
    - finding the number of linguistic variables, fuzzy terms (membership functions) in each, form of rules, etc.
  - parameter tuning
    - finding the exact values of the m.f. parameters, rule weights, defuzzification parameters, etc.
- All parts of the process can be parameterised and tuned
  - with the possibility of automatic methods

Selecting the 'Best'

- Usually, each candidate fuzzy model has associated with it some measure of how good it is
  - objective function
  - cost function
  - error measure, RMS error (RMSE), error
- The structure and parameters can be altered and the performance measure either maximised or minimised
  - maximise objective functions / performance
  - minimise cost functions / error

Direct Objective Function

- Consider the restaurant tipper
  - two inputs
    - (quality of) service
    - (quality of) food
  - one output
    - (amount of) tip
- In some situations, it might be a simple matter of maximising or minimising the output (tip)
  - maximise the tips given (good for waiters!)
  - minimise the tips given (good for customers!?)

Direct Cost Function

- It is more usual to have a fixed or pre-specified idea of the outputs desired
  - set the target output(s) for given input(s)

service	food	target tip
0	0	0
0	10	15
10	0	15
10	10	30
5	5	15

Direct Cost Function

- It is more usual to have a fixed or pre-specified idea of the outputs desired
  - set the target output(s) for given input(s)
  - calculate the root-mean-squared error

service	food	target tip	actual tip	error (= target-actual)	error^2
0	0	0	5.07	-5.07	25.7
0	10	15	15	0	0
10	0	15	15	0	0
10	10	30	24.92	5.07	25.7
5	5	15	15	0	0
				Σ	51.4
				Mean Squared Error $\Sigma/5$	10.3
Root-mean-squared error			(RMSE)	=sqrt(10.3)	3.2

Complex Example

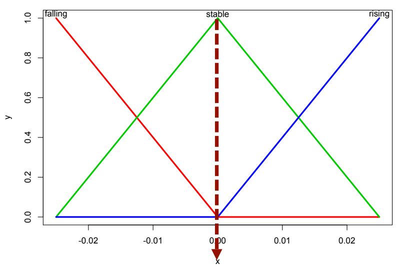
- We want to create a system to advise on buying or selling shares
  - inputs
    - FTSE index (UK stock market price)
    - pound / dollar exchange rate
      - actually the daily change in each of these
  - output
    - advice on whether to sell/hold/buy Microsoft shares
      - daily advice to sell/hold/buy

FIS Structure

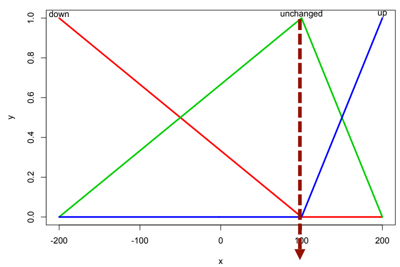
- Two inputs
  - FTSE index
    - three terms (m.f.s): falling, stable, rising
  - exchange_rate
    - three terms (m.f.s): down, unchanged, up
- One output
  - advice
    - three terms: sell, hold, buy
- Three rules
  1. If *exchange* is *falling* and *ftse* is *up* then *advice* is *buy*
  2. If *exchange* is *rising* and *ftse* is *down* then *advice* is *sell*
  3. If *exchange* is *stable* or *ftse* is *unchanged* then *advice* is *hold*

FIS Parameters

- Each of the two inputs has three triangular m.f.s with one shared control parameter



cp= 50
half-way along universe
(on a scale of 0 to 100)



cp= 75
three-quarters along
(on a scale of 0 to 100)

Indirect Objective Function

- Produces advice on buying/selling shares
  - there is no obvious error to minimise
    - we have no pre-specified target output
  - the output itself is not an objective function
    - we have no objective measure of goodness of advice
- A suitable *indirect objective function* can be created by using the advice to buy/sell shares
  - trade shares (unseen data) over a certain period
  - how much money does one end up with?

Fuzzy Model Example

```
evalmodel <- function(ms, out) {
  cash= 1000
  shares= 0

  n= length(out)

  for ( i in 1:n ) {
    if ( out[i] >= 55 && cash >= ms[i] ) {
        # buy
        shares= shares + 1
        cash= cash - ms[i]
    } else if ( out[i] <= 45 && shares > 0 ) {
        # sell
        shares= shares - 1
        cash= cash + ms[i]
    }
  }

  value= cash + shares * ms[i]
}
```

Search Methodologies

Exhaustive Search

- Evaluate each combination in turn using some systematic method, e.g.
  - start (1,1) → (1,99)
    (2,1) → (2,99)
    ...
    (99,1) → (99,99)
  - sometimes called the ‘brute-force’ approach
- This cannot be done for most real problems
  - there are too many combinations
  - too computationally expensive

Fuzzy Model Example

```
tune_ex <- function(inp, msr) {
    vmax= 0

    for ( i in 1:99 ) {
        for ( j in 1:99 ) {
            fis= makefis('shares', c(i, j) )
            out= evalfis(inp, fis)
            v= evalmodel(msr, out)
            if ( v > vmax ) {
                opt= fis
                vmax= v
            }
            rm(fis)
        }
    }

    return(list('fis'= opt,'vmax'= vmax))
}
```

Monte Carlo

- Suppose we just guess random set of parameters
- The Monte Carlo algorithm
  - generate a random starting position
  - evaluate the starting position and store it as best
  - repeat
    - generate a new random position
    - evaluate the new position
    - if the new position is better than the best found so far
      - store the new position as the best
  - until we decide to stop (e.g. not improved for 20 goes)
- This may or may not find the global maximum

Fuzzy Model Example

```
tune_mc <- function(inp, msr) {
    vmax= 0

    for ( L in 1:50 ) {
        cp= sample(99, 2, replace=T)
        fis= makefis('shares', cp)
        out= evalfis(inp, fis)
        v= evalmodel(msr, out)
        if ( v > vmax ) {
            vmax= v
            opt= fis
        }
        rm(fis)
    }

    return(list('fis'= opt,'vmax'= vmax))
}
```

Hill Climbing

- The hill climbing algorithm
  - start at either a fixed or random position
    - evaluate the current position
  - at each step
    - evaluate in each of the surrounding directions
      - up, down, left, right
    - move in direction of greatest improvement
  - stop if all moves are lower than current position
- This is guaranteed to find the peak (maximum)
  - but *only* if there is just one (global) maximum

Local Maxima

- Suppose the quality grid changes to this
- If the hill-climbing starts at e.g. (1,5) and heads *right* first
  - the global maximum at (5,5) is found
- However, if the hill-climbing starts at e.g. (1,5) and heads *up* first
  - a local maxima at (1,3) is found

flour

sugar	1	2	3	4	5	6	7	8	9
1	1	2	3	2	1	2	3	2	1
2	2	3	2	1	2	1	2	3	2
3	3	2	1	2	3	2	1	2	3
4	2	1	2	3	4	3	2	1	2
5	1	2	3	4	**5**	4	3	2	1
6	2	1	2	3	4	3	2	1	2
7	3	2	1	2	3	2	1	2	3
8	2	3	2	1	2	1	2	3	2
9	1	2	3	2	1	2	3	2	1

Stochastic Local Search

- The performance of local hill climbing is very dependent on the shape of the landscape
  - if the landscape has very many local maxima, the chance of finding the global maximum is small
- Why not combine Monte Carlo + Hill Climbing?
  - repeat
    - generate a random starting position
    - hill climb to the local maximum
    - store best local maximum
  - until *stopping_criteria*
- *Stochastic*: ‘random’
- *Local search*: moves in the local neighbourhood

Fuzzy Model Example

```
tune_hc <- function(inp, msr) {
      vmax= 0
      vt= rep(0,4)
      dp= rbind(c(-1,0),c(1,0),c(0,-1),c(0,1))

      for ( L in 1:20 ) {
          cp= sample(99, 2, replace=T)

          # hill climb to (local) maximum
          repeat {
              # get four surrounding positions (in random order)
              np= pmin(pmax(rep(cp,each=4) + dp[sample(4),], 1), 99)

              # evaluate each of the surrounding positions
              for ( k in 1:4 ) {
                  fis= makefis('shares', np[k,])
                  out= evalfis(inp, fis)
                  vt[k]= evalmodel(msr, out)
                  if ( vt[k] > vmax ) {
                     vmax= vt[k]
                     opt= fis
                  }
              }

              # stop if new best is not an improvement
              if ( (vnew= max(vt)) <= v )
                  break

              # make current position the best new one
              v= vnew
              cp= np[which.max(vt),]
          }
      }

      return(list('fis'= opt,'vmax'= vmax))
}
```

Simulated Annealing

Simulated Annealing

- Search algorithm inspired by physical annealing
  - adaptation of hill climbing which allows some downhill steps
    - accept all uphill steps
    - start by allowing all downhill steps and then gradually reduce the likelihood (depending on the size of the downhill step)
- In the *simulated annealing* algorithm
  - a temperature parameter controls the algorithm
  - initially, at high temperatures
    - all downhill steps are allowed
  - the temperature is gradually reduced
    - (step size / temperature) governs the chances of acceptance

SA Outline

- SA algorithm is usually implemented as a refinement of the basic hill climbing algorithm
  - initialise temperature, *T*
  - generate random solution, *i*
  - repeat
    - generate a new (nearby) solution, *j*
    - if *fitness*(*j*) is higher than *fitness*(*i*) then accept the move
    - if *fitness*(*j*) is lower than *fitness*(*i*) then
      - accept the move according to a probability which decreases with the size of the difference in fitness and increases with temperature, *T*
    - reduce the temperature, *T*
  - until stopping criteria

Downhill Steps

- The equation for accepting downhill steps comes from the physical annealing process

  $e^{-(\Delta E / T)}$
  - where ΔE is the *energy change*
- If the energy change is small (i.e. a small downhill move) or the temperature is high

  $e^{-(\Delta E / T)} \longrightarrow e^{-0} \approx 1$, move probably accepted
- If the energy change is large (i.e. a large downhill move) or the temperature is low

  $e^{-(\Delta E / T)} \longrightarrow e^{-\infty} \approx 0$, move probably rejected

SA Parameters

- Need to choose an initial temperature *T* such that
  - almost all moves are accepted (uphill & probably downhill)
  - the search is a 'random walk'
- The decrease the temperature *T* gradually, until
  - only uphill moves are accepted
  - the search is now a hill climb to the (local) optimum
- Can let the algorithm repeat a number of times at each *T*, rather than decreasing *T* after every step
  - the length parameter, *L*, specifies the number of steps to be repeated at each temperature, *T*

Fuzzy Model Example

```
tune_sa <- function(inp, msr) {
     vmax= 0
     cp= sample(99, 2, replace=T)
     dp= rbind(c(-1,0),c(1,0),c(0,-1),c(0,1))
     fis= makefis('shares', cp)
     out= evalfis(inp, fis)
     vmax= v= evalmodel(msr, out)

     Temp= 50
     for ( X in 1:50 ) {
         for ( L in 1:10 ) {
            np= pmin(pmax(cp + dp[sample(4,1),], 1), 99)
            fis= makefis('shares', np)
            out= evalfis(inp, fis)
            vnew= evalmodel(msr, out)
            if ( vnew >= v | runif(1) < exp((vnew-v) / Temp) ) {
              cp= np
              v= vnew
            }
            if ( v > vmax ) {
              vmax= v
              opt= fis
            }
            rm(fis)
        }
        Temp = Temp / 1.1
     }

     return(list('fis'= opt,'vmax'= vmax))
}
```

SA Problems

Real-Valued Parameters

- So far we have only considered integer values
  - only a fixed number of combinations of parameters
  - *combinatorial optimisation*
- Suppose one or more parameters are *reals*
  - what step length should be used in e.g. hill climbing?
- Too big?
  - might step right over a sharp peak
- Too small?
  - might spend too long searching uninteresting areas

The Solution

- Dynamic step length dependent on terrain
- Ideally using the shape of the terrain
- But gradients are not (usually) available
- Need a gradient-free dynamic algorithm
- Solution: Nelder-Mead simplex

  "A simplex method for function minimization"

  J.A. Nelder and R. Mead

  The Computer Journal

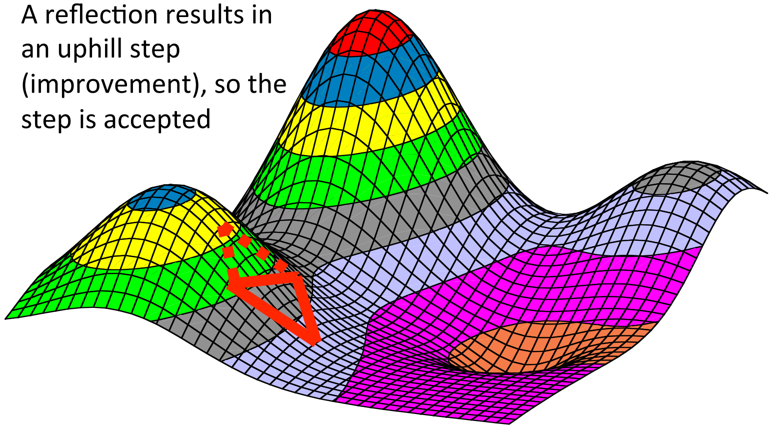
  Vol. 7: pp. 308–313, 1965

The Simplex Method

- A shape of with *N*+1 vertices is used where *N* is the dimensionality of the problem
  - e.g. a triangle in 2D
  - the shape starts at a random position
- Certain transformations of the shape are allowed
  - reflection of the lowest vertex through the opposite face
  - a reflection of the lowest with expansion
  - a contraction of the lowest towards the opposite face
  - a contraction along all faces towards the highest vertex
- If new point is higher, the new shape is accepted

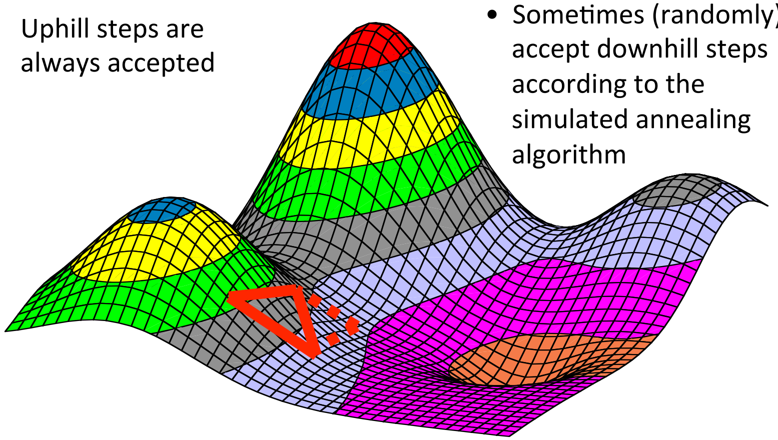
A Simplex Illustrated

- A reflection results in an uphill step (improvement), so the step is accepted



Simplex Method with SA

- Uphill steps are always accepted
- Sometimes (randomly) accept downhill steps according to the simulated annealing algorithm



Summary

- Lecture summary
  - fuzzy model identification comprises finding the structure and parameters of a fuzzy system
    - the distinction is arbitrary as both can be tuned
    - fuzzy model space is vast, so exhaustive search impractical
  - there are various (semi-) automatic algorithmic approaches that can be used to tune systems
    - in general, any optimisation method may be used
- Next lecture
  - ANFIS: adaptive neuro-fuzzy inference system