Overview

What is today about?

- Dealing with special data types
- What is optimisation and why do we need it?
- General-purpose optimisation techniques
- Time series data: prediction or simulation?
- Network
- Case study: fault diagnosis with network data

What is a “special” data type? What data do you have that is special?

Part I: Evolutionary Computing

Michael Small

⋆ Complex Systems Group
School of Mathematics and Statistics
The University of Western Australia

† Mineral Resources
Commonwealth Scientific and Industrial Research Organisation
Australia

Evolutionary Algorithms

- Computer algorithms “inspired” by natural processes:
  - artificial intelligence, cellular automata, sensor networks, excitable media, genetic algorithms
- Optimisation when differentiation (i.e. $\frac{d}{dt}$) is hard to do(objective function is not smooth)
  - genetic algorithms, ant colony optimisation, particle swarm optimisation, simulated annealing, tabu search, cultural adaption, ...

Discussion

What does optimisation mean for your industry?

- Objective function — the thing to be optimized $f(x)$ where we want to find x that makes $f(x)$ as *big/small* as possible
- Solution space $\mathcal{X}$ — the set of all possible solutions x

Genetic Algorithm

1. Choose a collection of possible solutions $\{x_1, x_2, x_3, \ldots\} \subset \mathcal{X}$
2. Evaluate the fitness $f(x_i)$ of each x_i
3. Breed them, mutate them and apply evolutionary pressure (kill off less fit) to get a group of children $\{y_1, y_2, y_3, \ldots\} \subset \mathcal{X}$ from the parents $\{x_1, x_2, x_3, \ldots\}$
4. Replace the parents with the children and repeat from 2

Genetic Algorithms

- Good for:
    1. difficult to evaluate objective functions
    2. high dimensional systems or non-smooth system
    3. discrete, categorical, or mixed variables
    4. paralellisation
    5. problems when we don't know much about the objective
    6. "modular" problems
- Bad for:
    1. smooth problems (use calculus)
    2. stochastic/noisy objective functions
    3. very high dimensional systems
    4. poorly parametrised problems (the genes don't work)
    5. binary objectives
    6. naiveté

Implementation

1. Define/determine objective function $f(x)$
   $\mathtt{f}(\mathtt{x}) = \frac{\cos\frac{1}{\mathtt{x}}}{\mathtt{x}}$
2. Encode $x \in \mathcal{X}$ as a gene consisting of building blocks
   $x \in [-1, 1]$ `where` $x = (-1)^{b_0} \times 0.b_1 b_2 b_3 \ldots b_{63}$ `is a sign bit and a sequence of 63 binary bits.`
3. Generate random initial population $\{x_1, x_2, x_3, \ldots\} \subset \mathcal{X}$
4. Compute the fitness $f(x_i)$ of each.
5. For each child y_i select two parents at random with probability proportional to $f(x_i)$ (their fitness) and perform cross-over mutation
6. Randomly mutate each gene of each child with probability m $(0 < m \ll 1)$.
7. Replace the parents with the offspring and repeat from 4

Pythonification

DIY is probably best, but if you insist:

1. GAFT (on github)
2. DEAP (on github)
3. pyevolve (on github)
4. Pyvolution (pypl.org)

Exercise

- Examine the provided code.
- What effect does mutation have on the speed of result?
- Modify to preserve the fittest
- 2D optimisation problem
- optimise networks and/or time series models

Reference

References

Clinton Sheppard “Genetic Algorithms with Python”.