The Flower Pollination Algorithm

Alexander G. Ororbia II
Biologically-Inspired Intelligent Systems
CSCI-633
4/16/2026

Flower Pollination (1)

- Over quarter million flowering plant species (80% plant species are flowering species), evolved for at least 125 million years

- Flower's main purpose = reproduction through pollination
  - Transfer of pollen, via pollinators such as insects, birds, bats, & others
  - Flower-pollinator relationship (some flowers can only attract/depend specific species of bird/insect)

Flower Pollination (2)

- <u>Pollination</u>: abiotic & biotic
  - ***Biotic*** = requires pollinators
    - Flower constancy (honeybees) – tend to visit certain flower species only, maximizes transfer of pollen in same/conspecific plants
    - Guarantees nectar supply for pollinators (given limited memory/learning)
  - ***Abiotic*** = does not require pollinators (wind/diffusion), e.g., grass
  - **Self-pollination**: pollination comes from same plant or its flowers, e.g., peach flower
  - **Cross-pollination**: pollination can come from pollen of different plant

The Flower Pollination Algorithm (FPA)

- Inspired by pollination process of flowering plants
  - Particularly useful for multi-objective optimization

- **<u>Four rules/concepts:</u>**
  1. Biotic & cross-pollination considered processes of global pollination, pollinators move according to *Lévy flight* / distribution (Rule 1)
  2. *Local pollination* – abiotic & self-pollination used (Rule 2)
  3. *Flower constancy* = reproduction probability proportional to similarity of 2 flowers involved (Rule 3)
  4. Interaction/switch between global & local pollination governed by *switch* probability $p \in [0,1]$, with slight bias towards local pollination (Rule 4)

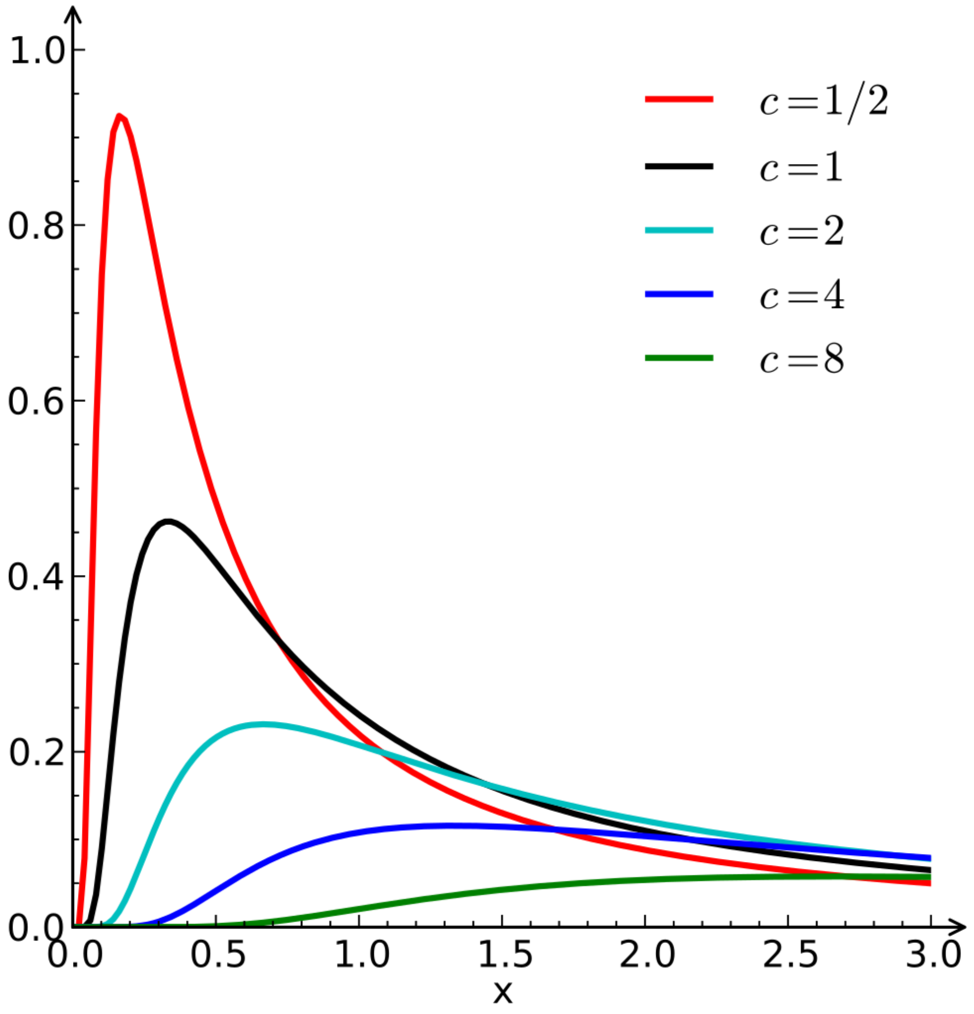
The Lévy Distribution

- Contus probability distribution for non-negative random variable (frequency is dependent variable)

$$f(x;\mu,c)=\sqrt{\frac{c}{2\pi}}\;\frac{e^{-\frac{c}{2(x-\mu)}}}{(x-\mu)^{3/2}}$$

- Book's variant for it:

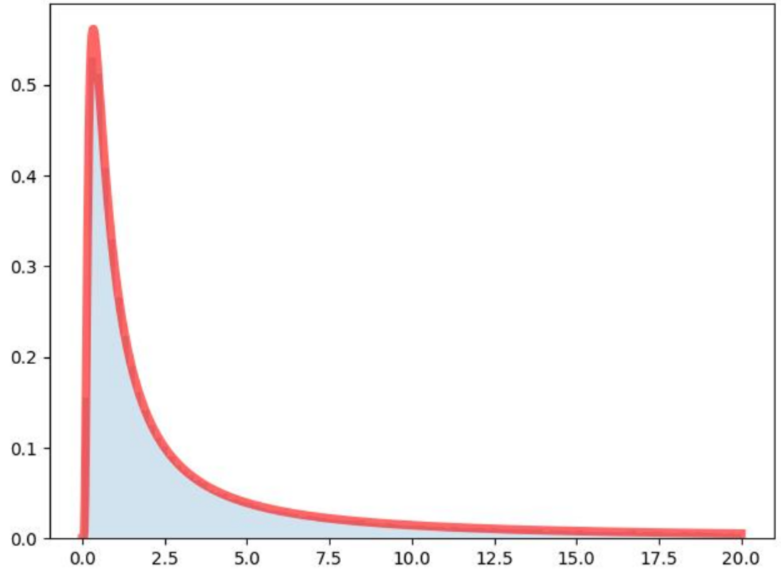
$$L \sim \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi}\frac{1}{s^{1+\lambda}}, \quad (s \gg s_0 > 0)$$



https://en.wikipedia.org/wiki/L%C3%A9vy_distribution

FPA Mechanics

```python
def sample_levy(u, c = 1.0, mu = 0.0):
    """
    Arguments:
        u: a uniform[0,1) random number
        c: scale parameter for Levy distribution (default: 1)
        mu: location parameter (offset) for Levy (default: 0)
    """
    return mu + c / (norm.ppf(1.0 - u)**2)
```



Flower Pollination Algorithm (or simply Flower Algorithm)

Objective min or max $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, x_2, ..., x_d)$
Initialize a population of n flowers/pollen gametes with random solutions
Find the best solution $\boldsymbol{g}_*$ in the initial population
Define a switch probability $p \in [0, 1]$
while (t <MaxGeneration)
  for $i = 1 : n$ (all n flowers in the population)
    if rand $< p$,
      Draw a (d-dimensional) step vector L from a Lévy distribution
      Global pollination via $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \gamma L(\boldsymbol{g}_* - \boldsymbol{x}_i^t)$
    else
      Draw ϵ from a uniform distribution in [0,1]
      Do local pollination via $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \epsilon(\boldsymbol{x}_j^t - \boldsymbol{x}_k^t)$
    end if
    Evaluate new solutions
    If new solutions are better, update them in the population
  end for
    Find the current best solution $\boldsymbol{g}_*$
end while
Output the best solution found

The Real World Contains Multiple Objectives

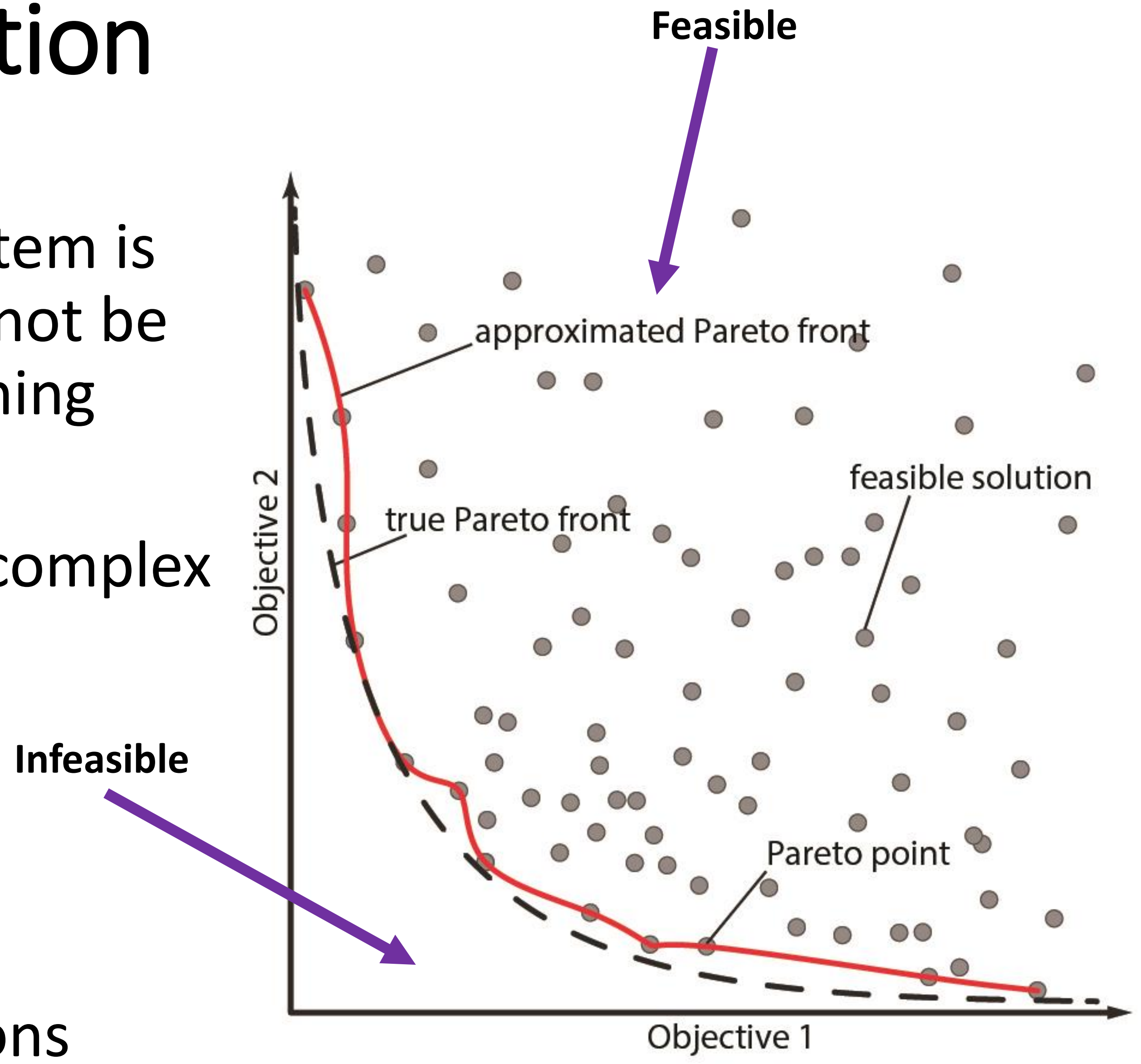
- Want to improve performance of product while minimizing cost at same time
- ***Problem***: multiple objectives may be conflicting -- trade-offs are required to reach a “balance”
  - This means multiple algorithmic runs are needed to generate a set of solutions
  - Requires good approximations of **Pareto fronts** for problem of interest

Multi-Objective Optimization

- **Pareto Optimality** = state in which system is optimized such that one objective cannot be improved without another one worsening

- Number of objectives results in more complex Pareto fronts
  - Single objective = point
  - Bi-objective = curve
  - Tri-objective = surface

- **<u>Goal</u>**: search for Pareto optimal solutions



https://iam.tugraz.at/augmented-parametrics/augp-control-system/

Applications of the Multi-Objective Flower Pollination Algorithm (MO-FPA)

- Spring design optimization (single objective)
- Welded beam design (single objective)
- Pressure vessel design (single obj.)
- Disc brake design (multi obj.)

- MO-FPA generally strong in approximating Pareto fronts yet…
    - …MO-FPA behavior is hard to analyze (related to central problems of FPA's dynamics as discussed earlier)

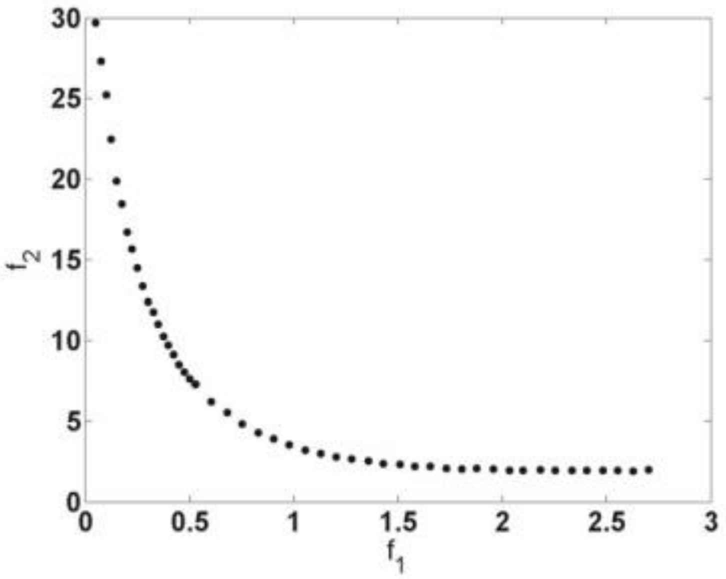


Figure 11.3 Pareto front of the disc brake design.

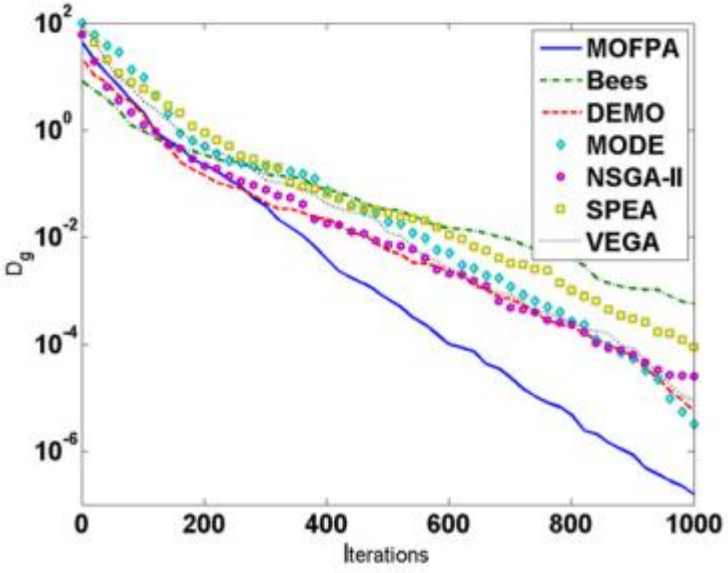
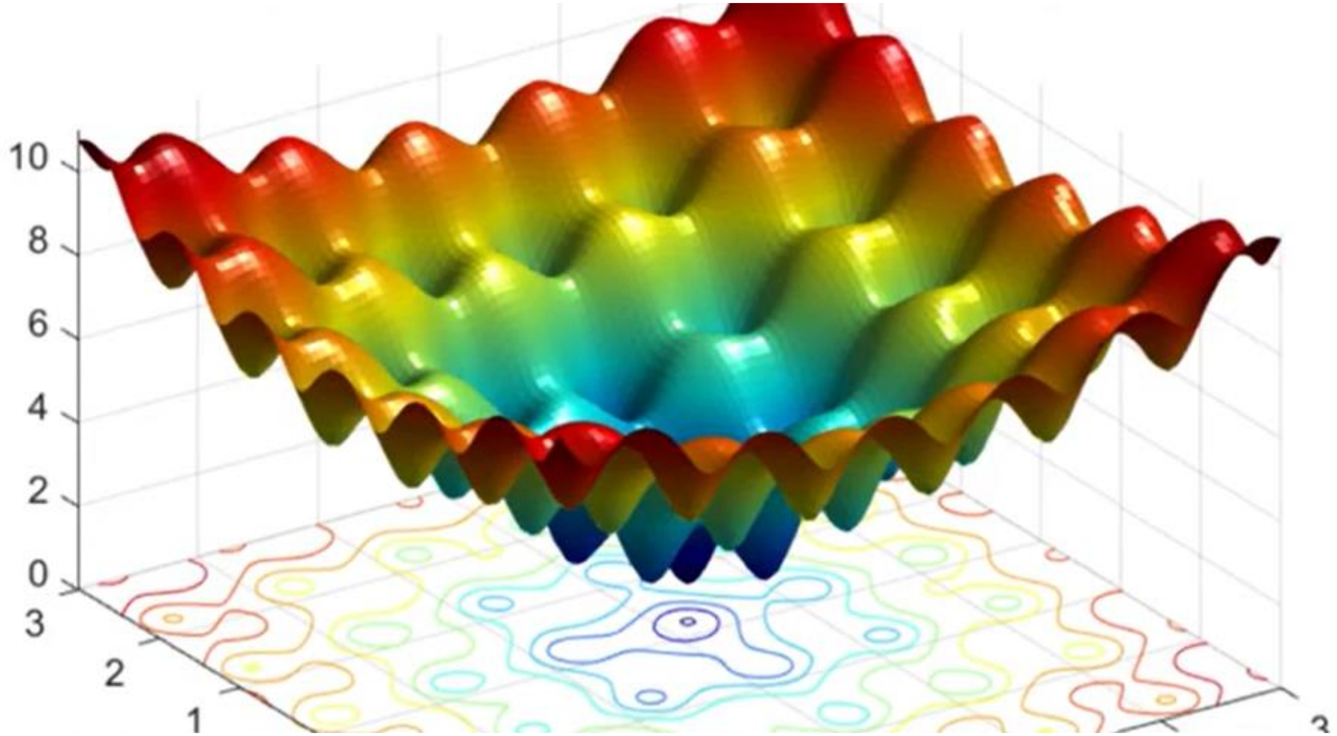


Figure 11.4 Convergence comparison for the disc brake design.

FPA in Action

- https://www.youtube.com/watch?v=oZzUPspIrUg



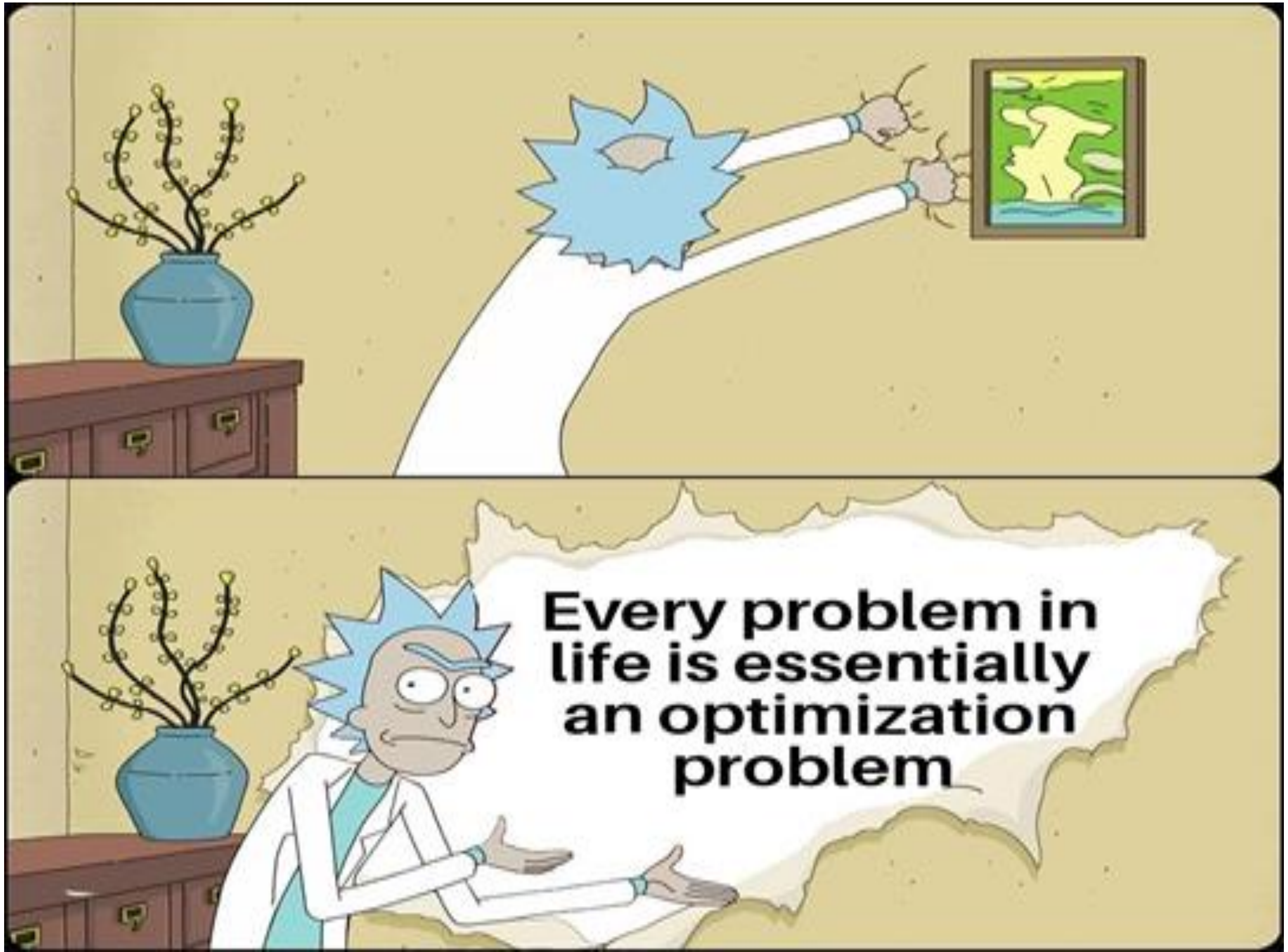
FPA Works Pretty Well And…

- It's very efficient
  (near exponential convergence rate in comparison to other algo's)

- It only only has two hyper-parameters ρ and γ
  - Yet difficulty of nonlinear flight distribution makes analysis difficult (main update can be analyzed via dynamical systems or Markov chains)
  - Work needed to combine FPA w/ other methods to better handle constraints

Questions?

Questions?


Every problem in life is essentially an optimization problem

Step 1. Initialization: pollen population $x = (x_1, x_2, .., x_d)$ is generated randomly. A switch probability $p \in$ arrange from 0 to 1. A stopping criterion is set.

Step 2. The best solution g^* is calculated with initial population, F_{min} is assigned to fitness at g^*.

Step 3. For each pollen in the population

if rand < p,

  A step vector L is computed as Lévy distribution Eq.(2)

  Global pollination is updated via Eq.(1)

else

  Draw u from a uniform distribution in [0,1]

  Local pollination is processed via Eq.(3)

end if

Step 4. Evaluate new solutions, the function value F_{new} is assigned to fitness(x $(t+1)$). A new solution is accepted if the solution improves (F_{new} less than F_{min}), by updating the best solution g^* to $x(t+1)$ and assign the minimum function F_{min} to F_{new}.

Step 5. If the termination condition is not safety, go to Step 3.

Step 6. Output the best solution found.