

Computational Systems Biology
Karthik Raman
Department of Biotechnology
Indian Institute of Technology - Madras


Lecture – 48
PyGMO

(Refer Slide Time: 00:11)

Computational Systems Biology
PyGMO

- ▶ Introduction to PyGMO
- ▶ PyGMO Algorithms and Examples
- ▶ Multi-objective Optimisation

Karthik Raman
Department of Biotechnology, Bhupat & Jyoti Mehta School of Biosciences
Initiative for Biological Systems Engineering (IBSE)
Robert Bosch Centre for Data Science and Artificial Intelligence (RBC DSAI)
INDIAN INSTITUTE OF TECHNOLOGY MADRAS






In today's video, I will introduce you to this very interesting software library called PyGMO. It is written in python and it is, it also interfaces with some very efficient C code which is why it is essentially a python wrapper for some C code. So it is very fast and I will be taking about some of the algorithms implemented in python, in python GMO and PyGMO and some examples as to how you go about using this library.

You will be interested to see that there are so many different kinds of flavours of evolutionary algorithms that are readily available in PyGMO which you can use immediately and I will also introduce you to the concept of multiobjective optimization. So what you do when you have more than 1 object to worry about. For example, in our case, it could be more than data from more than 1 experiment. How do you estimate parameters? I have told you different algorithms but what tools do you use, let us look at a very powerful tool called PyGMO.


(Refer Slide Time: 01:06)

OVERVIEW

<http://esca.github.io/pygmo/>


- **PyGMO: Python Parallel Global Multiobjective Optimizer**
- A scientific library providing a large number of optimisation algorithms (and problems!)
- Builds on PaGMO: Initially developed within the European Space Agency
 - Code was intended to help the automated design of interplanetary trajectories and spacecraft transfers in general




So PyGMO is basically a python package. It stands for Python Global, Python Parallel Global Multiobjective Optimizer. It is a scientific library which provides a large number of optimization algorithms as well as problems. There are many test problems that people use to stretch test any given algorithm. There is something known as the Rosenbrock function and things like that. This builds on PaGMO which was initially developed within the European Space Agency for, you know, design or interplanetary trajectories and spacecraft transfers and Mangalyan stuff, right.

(Refer Slide Time: 01:41)

OVERVIEW



- Powerful parallelization abstraction built around the generalised island-model paradigm
- All algorithms are automatically parallelised to leverage multiple cores
 - Implements a generalized migration operator: allows the user to easily define "migration paths" (topologies) between a large number of "islands" (CPU cores)
- Has efficient implementations of state-of-the-art bio-inspired algorithms
- **PyGMO can be used to solve constrained, unconstrained, single objective, multiple objective, continuous, mixed integer optimisation problems**

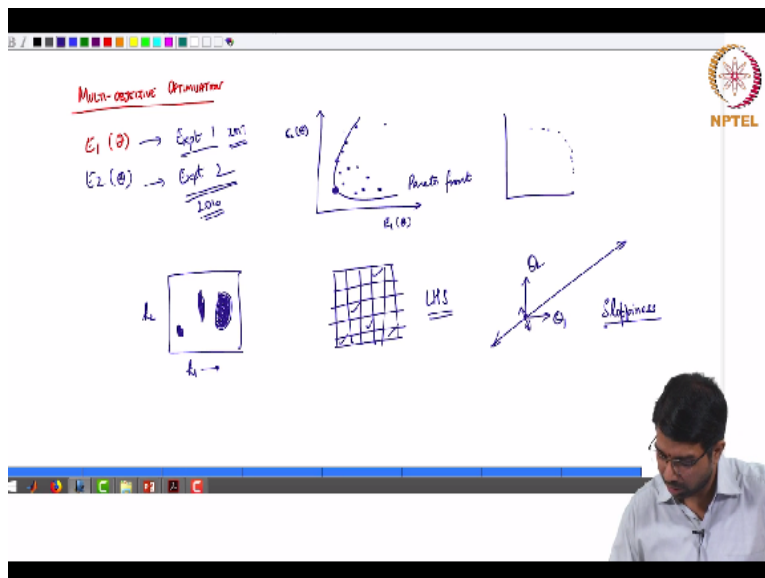


So it is a powerful parallelization abstract built around the general island model paradigm. So we will look at what this is? So all algorithms are automatically parallelized to leverage multiple cores, right. So they are, they work in basically islands. So there is a generalized migration

operator which allows the user define migration paths between these parallel optimizers. So it is, it is to say in some sense, let us say you have 8 CPU cores, you can run simulated annealing on one, DE on one, EA on 5 others, shuffle the solutions after a few iterations, continue, right.

This is as aggressive an optimization strategy as we can imagine, right. You are trying to leverage the best of all worlds, right. There is some strength for EA, there is some strength for simulated annealing, there is some strength for particles from optimization. You mix and match all of these in whatever way you want, right. So PyGMO can be used to solve constrained, unconstrained, single objective, multiobjective, continuous and mixed optimization problems.

(Refer Slide Time: 02:50)



A slight detour, what do we mean by multiobjective optimization. What is multi-objective optimization? **“Professor - student conversation starts”** (03:12) **“Professor - student conversation ends.”**

Meaning you have more than 1... Cost function. Cost function, right. So you may have E_1 of theta and you may have E_2 of theta. There are 2 different cost functions. Maybe this comes from experiment 1, this comes from experiment 2, right and you may want to optimize, jointly optimize them or separately optimize them but the moment you have 2 objective functions, how do you say what is the best solution?

Because they have 1 objective function, I would say give me the theta that minimizes E1, E of theta. Now the one that gives you low E1, may give you high E2 and vice versa. So you will have to start worrying about it. So typical way is to plot it, right. You may get something like this and you may end up computing what is known as the Pareto front.

“Professor - student conversation starts” Sir, instead why cannot we multiple with E1 by some scalar and... You can, you can add E1 and E2 if you want, right and minimize it. That is a possibility but usually these are 2, you can multiply them, you can take, you can take some function of E1 and E2 obviously. But you may want to worry about, you may want to jointly optimize the 2, right. So here, if at this point, you know that, you know, you got a very good value for both E1 and E2, right.

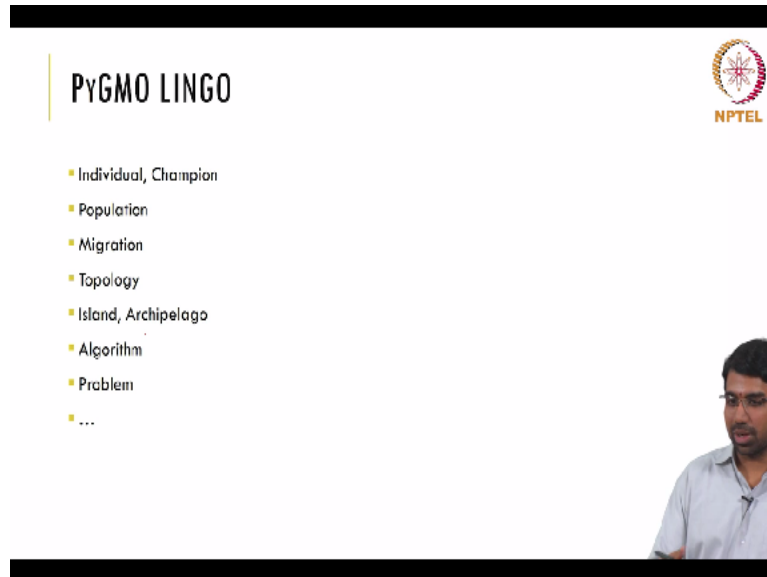
So if you go in this direction, you will increase E2. If you go in this direction, you will increase E1 but at this point, you have seemingly very good E1 and E2 values. Or you know, you would have, like the front that looks more like this. You can have different kinds of fronts basically. So if you have a different front, the optimization might look harder, right. What if the points were all, you know, distributed like this, right?

It becomes harder. So, but you can basically, you may have to judge. You may have to judge which is the more important experiment to satisfy, right. Because they may have not come from the same labs. You may, or this is an experiment that was reported in, you know, 2010 and this is an experiment which was reported in 2017, right. You may want to make sure that you stick more with this, may be the methodologies are slightly different, more in accuracy is here, whatever, right or more inaccuracy is here, whatever, right.

You may have to bring in subjectivity, right. You cannot objectively say that yes just use some $E1+E2$ or things like that. Sir for a given parameter set, you are finding the best cost function, right. No, for a given cost function, we are trying to find the best parameter set. Okay. Right. For a given cost function, we try to find the best parameter set. **“Professor - student conversation ends.”**

So it gives a multiobjective, continuous, even mixed integer optimization problems. What are mixed integer problems? Some solutions are integers. They are numbers, right. You cannot have 1.3, it is not continuous.

(Refer Slide Time: 07:28)



So there are, there is a lingo for PyGMO. There are individuals, champions, populations, migrations, topology, island, archipelagos, algorithms and problems, right. I think most of them are self-explanatory. I think you will care to understand what individual is. Champion is the best of the individuals in a given round. Population, yes of course.

Migration is migration between different islands in an archipelago, right. So you set up multiple islands of optimizers and each island will have an algorithm which will run on a particular problem. You need to create a problem. This is basically object oriented programming stuff. So you need to define a problem, class and so on and so forth.

(Refer Slide Time: 08:14)

ALGORITHMS



Algorithm	PyGMO module	Algorithm	PyGMO module
Differential Evolution (DE)	de	S-Metric Selection (MOA) (SMS-EMOA)	sms_emoa
Self-adaptive DE (jDE)	jde	Corana's Simulated Annealing (SA)	sa_corana
DE with p-best crossover (mde_pbx)	mde_pbx	Parallel Decomposition (PADE)	pace
Differential Evolution (DE)	de_1220	Non-dominated Sorting PSO (NSPSO)	nspsa
Particle Swarm Optimization (PSO)	pso	Strength Pareto EA 2 (SPEA2)	spea2
Particle Swarm Optimization (PSO)	pso_gen	Artificial Bee Colony (ABC)	bee_colony
Simple Genetic Algorithm (SGA)	sga_gray	Improved Harmony Search (IHS)	ihs
Simple Genetic Algorithm (SGA)	sga	Monte Carlo Search (MC)	monte_carlo
Vector Evaluated Genetic Algorithm (VEGA)	vega	Monte Carlo Search (MC)	py_example
(N+1) EA Evol. Algorithm (SEA)	sea	Covariance Matrix Adaptation ES	py_cmaes
Non-dominated Sorting GA (NSGA2)	nsga_11	Covariance Matrix Adaptation ES	cmaes



What are the algorithms? Nice assortment of algorithms. You have differential evolution, self-adaptive DE, DE with p-best crossover. This is something we found to be very useful when we studied a few bio-models. Of course, particle, particle swarm and swarm optimization and simple genetic algorithm and N+1 evolutionary algorithm or evolutionary strategy, so on and so forth. Monte Carlo search, CMA covariance matrix adaptation in evolution strategy and so on.

These are all popular algorithms and it also has a particular flavour of simulated annealing called Corana's simulated annealing algorithm. There are bunch of algorithms that are available.

(Refer Slide Time: 09:03)

PYGMO EXAMPLE



```
from PyGMO import *
prob = problem.schwefel(dim = 50)

algo = algorithm.de(gen = 500)
isl = island(algo,prob,20)
print isl.population.champion.f
isl.evolve(10)
print isl.population.champion.f
```



And as simple as this to do PyGMO and of course you need to be familiar with python but if you

are familiar with python, it basically is just from PyGMO import everything and problem, this is a particular problem that is defined in PyGMO. This has to be replaced with your problem. You need to set up your parameter optimization problem here, right and algo is algorithm. Differential evolution for 500 generations, create 20 islands, print island population champion function value.

What is the function value of the champion in the population in all islands, right? So this is basically the initial value. This is essentially saying print X0. Now you evolve it for 10 generations or longer and you then, you evolve it 10 times and then you, you print the champion across all those epochs.

(Refer Slide Time: 10:06)

The slide is titled "WHAT CAN PYGMO DO?". It features the NPTEL logo in the top right corner. On the left, there are three bullet points: "Can easily scale to multiple CPUs", "Can easily migrate solutions...", and "... even between algorithms!". To the right of these points is a code block containing Python code for setting up a PyGMO archipelago. Below the code is a small red diagram of a ring topology. In the bottom right corner of the slide, there is a video feed of a man speaking.

WHAT CAN PYGMO DO?

- Can easily scale to multiple CPUs
- Can easily migrate solutions...
- ... even between algorithms!

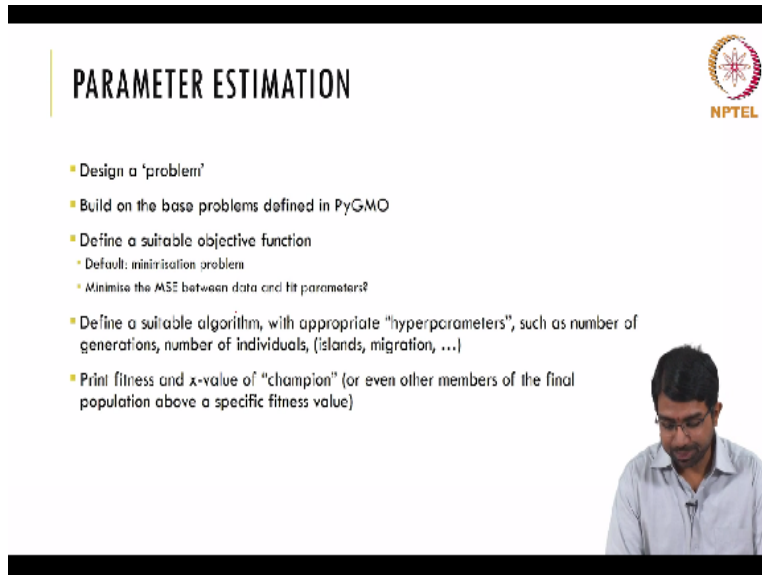
```
from PyGMO import *
prob = problem.scheffcl(dim = 50)
algo = []
for i in range(1,9):
    algo.append(algorithm.de(gen=500,variant=i))
archi = archipelago(topo-topology.ring())
for i in range(0,8):
    archi.push_back(island(algo[i],prob,20))
print min([isl.population.champion.f for isl in archi])

archi.evolve(20)
print min([isl.population.champion.f for isl in archi])
```

So what can PyGMO do? It can scale easily to multi CPUs. You can migrate solutions even between algorithms. As I said you can have DE, PDE (()) (10:16) and just throw solutions from one to the other. So, so you create an archipelago with a ring topology meaning... and each of this (()) (10:03) with an algorithm, right. So there are i variants for differential evolution. So you could variant 1 here, variant 2 here, variant 3 here, 4, 5, 6, evolve them for a while.

And then allow the algorithms to transfer the champions, transfer the information continue to evolve. So you can see that this is a really aggressive approach to parameter optimization, right and you need to be this aggressive if you want to solve some really complicated complex optimization problems.

(Refer Slide Time: 11:07)



The slide is titled "PARAMETER ESTIMATION" and features the NPTEL logo in the top right corner. It contains a list of five bullet points:

- Design a 'problem'
- Build on the base problems defined in PyGMO
- Define a suitable objective function
 - Default: minimisation problem
 - Minimise the MSE between data and fit parameters?
- Define a suitable algorithm, with appropriate "hyperparameters", such as number of generations, number of individuals, (islands, migration, ...)
- Print fitness and x-value of "champion" (or even other members of the final population above a specific fitness value)

In the bottom right corner of the slide, there is a small video inset showing a man with glasses and a light-colored shirt looking down.

So how do we go about in the case of biological problems? You design a problem. There are base problems defined in PyGMO. So you build on that. In classic object oriented fashion, you extend those definitions. The default is the minimization problem, may be you can minimize the mean square error between data and fit parameters or use whatever objective function we have been talking about all along.

Define a suitable algorithm with appropriate hyperparameters such as generations, number of individuals, etc, etc and then print the fitness and function, the parameter value of the champion or even other members of the final population above a fitness value. So anything that is more than, less than 5% error, you can print those parameters and then you can study these parameter spaces. We will come to that in a moment, right.

(Refer Slide Time: 11:55)

USING MULTIPLE CORES



- Extremely easy!
- `archi.evolve(10)` evolves 10 parallel populations across available cores...

```
from PyGMO import *
prob = problem.schaeffel(dim = 50)
algo = algorithm.de(gen=100)
archi = archipelago(algo, prob, 8, 20)
print min([isl.population.champion.f for isl in archi])
archi.evolve(10)
print min([isl.population.champion.f for isl in archi])
```

- One can also look at all the final populations and pick "good" ones out of them, instead of picking just the best

Check out:
<http://esa.github.io/pygmo/documentation/island.htm>



And supertrivial to use multiple course, right. This automatically evolves across 10 parallel course, finished. No work at all, right and please check out the documentation here.

(Refer Slide Time: 12:17)

OTHER TOOLS



- LibRoadrunner gives good helper functions for loading SBML models and simulation etc. *SBML*
- As easy as:

```
import roadrunner
rr = roadrunner.RoadRunner("mymodel.xml")
result = rr.simulate()
rr.plot()
```



The other tool called libRoadrunner, this helps you directly connect with SBML, right. So all you have to do is import roadrunner, roadrunner. roadrunner mymodel.xml, simulate, plot. So this is where all your integration automatically happens. You want to connect this to PyGMO. Very well. So one interesting thing that remains to be discussed is we did see it briefly yesterday, right.

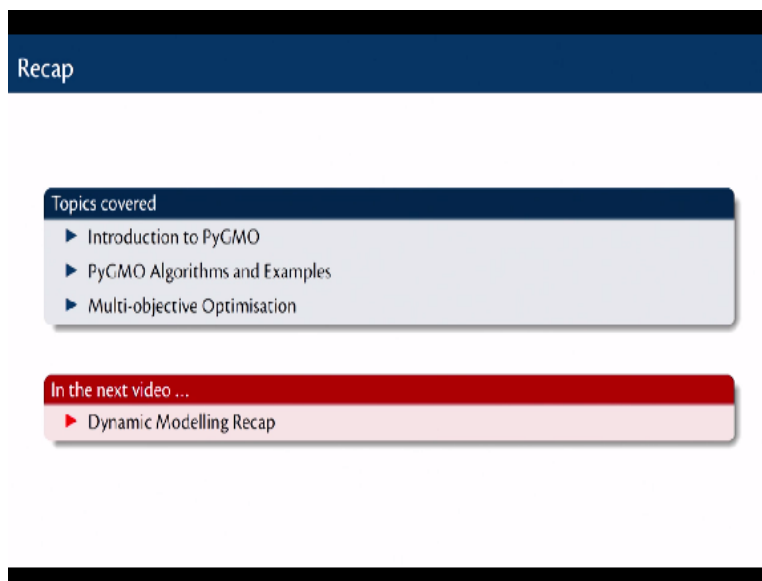
You have such parameter spaces. How do you study them, right? Or how do you even find these parameter spaces? So you may want to systematically uniformly sample across this using latin

hypercube sampling on one of those methods. You can read more about latin hypercube sampling. There are many sampling strategies. So you may want to sample like this and figure out how the space looks like.

One very interesting aspect is you may find that, let us say these are 2 parameter directions, theta 1 and theta 2. You may find that your cost function varies a lot in this direction but relatively very little in this direction, right. So you will have to start worrying about something known as sloppiness. The sloppiness is a very important issue in biological systems. It has fundamental implications for how easily you can estimate parameters or with what reliability you can fit parameters.

It depends upon the model to some extent on the data. It is quite complex but is a very interesting topic.

(Refer Slide Time: 14:39)



So in today's video, I hope you got a nice introduction to the very interesting software library API called PyGMO. It is written in python but it is very efficient and very useful for performing all kinds of, you know, GAs, right or evolutionary algorithms in general. So be it differential evolution or genetic algorithms or evolution strategies and, or even simulated annealing and so on.

And we looked at what are all the different algorithms that are implemented in PyGMO and a few examples, code snippets, to tell you how you go about writing PyGMO code and I also introduced you to this very interesting concept of multiobjective optimization and you should read more about it because I only gave you a very brief introduction. In the next video, we are essentially coming to the end of that big modelling. So I will give you a recap of the entire series of lectures we have had on dynamic modelling.