Solving Global Optimisation Problems

Johan Blåbäck

Università di Roma
"Tor Vergata"

Loosely based on techniques used in:

1912.XXXXX
with: Iosif Bena, Mariana Graña, Severin Lüst

1810.11365, 1310.8300, 1301.7073
with: Ulf Danielsson, Giuseppe Dibitetto

1312.5328
with: Diederik Roest, Ivonne Zavala

https://gitlab.com/johanbluecreek/pres-solving_global

@ de Sitter constructions in String Theory / IPhT CEA/Saclay
2019-12-11

What is an "optimisation problem"?

Given a function

$$\texttt{fitness}\colon \mathbb{R}^n \mapsto \mathbb{R}\,, \tag{1}$$

we want to find optima (minima) to this function.[1]

If you have gradients, are satisfied with local optima there are a number of algorithms that given a starting position x returns the closest optima x'.

What can you do if you are looking for a global optima?

What if you have no gradients or even discontinuities?

[1] As physicists we may call this function a scalar potential in other contexts it is called a cost-function.

The following are example optimisation problems we have.

de Sitter optimisation problem

- Solve equations of motion
- Positive potential
- Positive moduli masses
- Weak coupling, large volume
- Other hierarchies

Quintessence optimisation problem

- Keeping potential positive
- Minimise $\epsilon := \frac{1}{2}\frac{|DV|^2}{V^2}$
- Minimise $\eta := \min(\mathrm{eigval}(DDV))/|V|$
- Other wishes

Which we are not always able to approach analytically.

One CS subject that attack these problems is evolutionary computing[2]

$$\text{Computer science} \supset \text{artificial intelligence} \supset \text{evolutionary computing}$$

Evolutionary computing is population based and sometimes incorporates the following evolutionary operations

- Mutation
  Perturb candidates slightly
- Crossover
  Have two or more candidates interact and exchange "genetic data"
- Selection
  Have some selection method for singling out the candidates with better fitness

In our example optimisation problems our candidates are something like

$$[\text{fluxes} \ldots, \text{moduli} \ldots] \tag{2}$$

[2] Generally distinct from "machine learning".

How would you implement one?

Wikipedia[3] gives us a list

- Ant colony optimization
- Artificial life (also see digital organism)
- Differential evolution
- Estimation of distribution algorithms
- Evolutionary programming
- Gene expression programming
- Genetic programming
- Learnable evolution model
- Memetic algorithms
- Particle swarm optimization
- Self-organization
- Artificial immune systems
- Cultural algorithms
- Dual-phase evolution
- Evolutionary algorithms
- Evolution strategy
- Genetic algorithm[4]
- Grammatical evolution
- Learning classifier systems
- Neuroevolution
- Synergistic Fibroblast Optimization
- Swarm intelligence

Lets select differential evolution.

[3] https://en.wikipedia.org/wiki/Evolutionary_computation

[4] 1907.10072: Cole, Schachner, Shiu

Differential Evolution

Mutation

Select a candidate, x, and one random pair of other candidates: y and z. The candidate x is mutated as

$$x' = x + F(y - z)\,; \qquad F > 0 \tag{3}$$

Crossover

Crossover selects some entries of x' instead of x, e.g. randomly, to form x''

$$x''_i = x'_i \texttt{ if rand() } < C_r \texttt{ else } x_i\,; \qquad 0 < C_r < 1 \tag{4}$$

Selection

Then x or x'' is selected for the next generation via e.g. a greedy choice

$$x'' \texttt{ if fitness}(x'') \texttt{ < fitness}(x) \texttt{ else } x \tag{5}$$

DE/rand/1/bin

Now you could implement this yourselves. But you don't have to.

`BlackBoxOptim.jl`[5] (BBO) is an implementation in the Julia programming language.[6]

A number of global optimisation algorithms are implemented in BBO:

Natural Evolution Sampling	3
Differential Evolution Optimizers	5
Generating Set Search	2
Resampling Memetic Search	2
Stochastic Approximation	1
Random Search	1

[5] Robert Feldt, https://github.com/robertfeldt/BlackBoxOptim.jl

[6] 1411.1607: Bezanson, Edelman, Karpinski, Shah

Basic usage:

```
bboptimize(fitness; MaxTime = [time], PopulationSize = [size],
                    SearchSpace = [space], ...)
```

Initialises a population randomly distributed, and applies e.g. differential evolution for a fixed time or number of steps.

Returns an object containing all candidates of the final population and their fitnesses.

Just to demonstrate, lets take a concrete problem.

Kallosh, Wrase[7] used a nilpotent field to incorporate the addition of anti-branes in e.g. the "STU"-model. More specifically here: type IIA on $T^6/(\mathbb{Z}_2 \times \mathbb{Z}_2)$ + metric fluxes $\equiv S^3 \times S^3/(\mathbb{Z}_2 \times \mathbb{Z}_2)$ (isotropic). (Timm's talk)

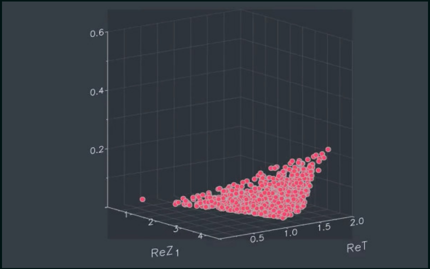
Following this paper[8] we can solve the equations of motion analytically, and we can filter out points that has $V > 0$ and $\eta > 0$. This they also do.

There are two branches of solutions to the e.o.m.s parametrised by $\operatorname{Re} Z_1$, $\operatorname{Re} Z_2$, and $\operatorname{Re} T$; 3-dimensional problem.

[7] 1808.09427: Kallosh, Wrase

[8] 1811.07880: Banlaki, Chowdhury, Roupec, Wrase

Random points looks like[9]



The problem with this model, without additions, is that $\tau^4 = \mathrm{Re}Z_1\mathrm{Re}Z_2^3$ and $\rho = \mathrm{Re}T$ are not $\gg 1$. How far can you push it?

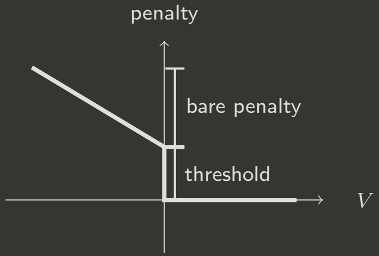
[9] 1811.07880: Banlaki, Chowdhury, Roupec, Wrase

To apply BBO to answer this question we must design the fitness function.

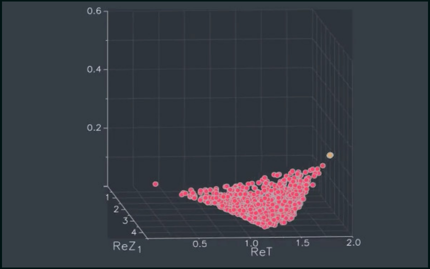
We want our fitness to solve the following:

- Positive potential
- Positive masses
- Minimize $1/\rho$ and $1/\tau$

We solve all by form a weighted sum of penalties.



From there it takes “a little bit of programming”,[10] and you can get your result.



The optima found is $\rho \approx 1.79$ and $\tau \approx 0.554$.

[10] https://gitlab.com/johanbluecreek/pres-solving_global

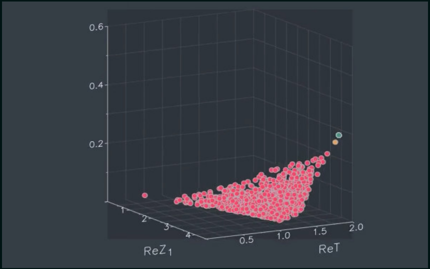
You may then ask: How much further can we go if we relax our demands and look for quintessence?

We no-longer solve equations of motion, hence the search space is 9-dimensional.

The fitness function is a weighted sum of penalties for:

- Positive potential
- $\epsilon > 0.1$
- $|\eta| > 0.1$
- Minimize $1/\rho$ and $1/\tau$

"A little bit of programming",[11] again gives us some result.



The optima found is $\rho \approx 1.82$ (3%) and $\tau \approx 0.594$ (7%).

[11] https://gitlab.com/johanbluecreek/pres-solving_global

The point here is not some weak attempt at showing off.

I want you to be able to not only understand, but also do these things too.

My problem is then: You may not be very willing to learn Julia just like that.

How do I convince you?

Just imagine all the programming you would have to do:

- Save searches
- Load old searches and continue
- Log searches to see what the results were
- Know what you saved and how your code changed from that point
- Some easy way to print old results if you change the code

After all that then you would have to design your fitness function.

Just imagine all the programming you would have to do:

- Save searches
- Load old searches and continue
- Log searches to see what the results were
- Know what you saved and how your code changed from that point
- Some easy way to print old results if you change the code

After all that then you would have to design your fitness function.

Good thing I solved that for you.

`bbsearch.jl`[12] is what I call the project where you can use BBO and minimize the amount of fuzz.

- Save searches
- Load old searches and continue
- Log searches to see what the results were
- Keeps track of your code (using `git`)
- You can print results of old searches

Most of the work is now designing a fitness function and executing:

```
$ ./bbsearch.jl -n -s problem_name -r 60
```

Now you say: "Hol'up, you make it sound too easy; Symbolic expressions?"

[12] `bbsearch.jl: https://gitlab.com/johanbluecreek/bbsearch`

This little magic function solves that for you:

```
function buildfun(name::String, expr_file::String, vars::String)
  expr = open(expr_file) do file
      read(file, String)
  end
  fun_string = """
  begin
    function $name($vars)
      $expr
    end
    $name(x::Array{Float64,1}) = $name(x...)
    $name(x::Vector) = $name(x...)
  end
  """
  eval(Meta.parse(fun_string))
end
```

```
buildfun("V","V.txt","ReT, ReZ1, ReZ2, h");
V(1.7911, 4.42381, 0.276983, 4.49756)
```

You can apply this to many other problems.

F-theory on $K3 \times K3$: The intersection matrix d and the flux matrix G (22×22) determine the size of the tadpole and stability. (Severin's Talk)

We[13] construct penalties for: $N = GdG^T d$

- No negative eigenvalues for N
- No imaginary parts in eigenvalues for N
- All zero eigenvalues except one (a_i)
- All non-distinct eigenvalues (between a_i and b_k)
- Minimize $\mathrm{Tr}\, N - 48$

[13] 1912.XXXXX: Bena, JB, Graña, S. Lüst

Summary

Algorithms from Evolutionary Computing are intended to solve global optimisation problems.

My intention here was to show

- how these strategies work (Differential Evolution in particular)
- how you can start using them (`BlackBoxOptim.jl` and `bbsearch.jl`)
- strategies for developing your own fitness functions

There are probably many problems that are considered “hard” when they really should not be. The approaches can help you.

Thank you for your attention.