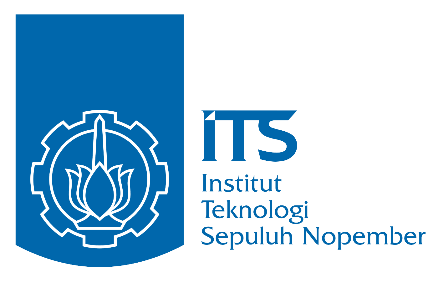


|  |
| --- |
| LAPORAN  UTS – TOPIK KHUSUS (B)  PARTICLE SWARM OPTIMIZATION |
|  |
| 1 november 2018  TEKNIK INFORMATIKA - ITS  Oleh: Son Ardhynata Sukarno Mudha NRP: 05111340000107 |



## Pernyataan

“Demi Allah Yang Maha Esa, maka dengan ini, saya bersumpah dan menyatakan dengan sebenar-benarnya bahwa saya mengerjakan jawaban soal Ujian Tengah Semester (UTS) ini secara sendiri dan mandiri, tidak melakukan kecurangan dalam bentuk apa pun, tidak menyalin/menjiplak/melakukan plagiat pekerjaan/karya orang lain, serta tidak menerima bantuan pengerjaan dalam bentuk apa pun dari orang lain. Saya bersedia menerima semua konsekuensi dalam bentuk apa pun, apabila saya ternyata terbukti melakukan kecurangan dan/atau penyalinan/penjiplakan/plagiat pekerjaan/karya orang lain.”

Surabaya, 1 November 2018



Son Ardhynata Sukarno Mudha

05111340000107

## Latar Belakang

Particle Swarm Optimization (PSO) adalah metode optimisasi stokastik berdasar populasi (*population-based*) yang diinspirasi dari tingkah laku sosial makhluk hidup yang sering hidup secara berkelompok seperti burung dan ikan laut. Metode ini diperkenalkan oleh Dr. Eberhart dan Dr. Kennedy pada tahun 1995.

PSO memiliki kesamaan dengan teknik komputasi evolusi (*evolutionary computation techniques*) seperti Genetic Algorithm (GA) seperti menginisialisasi populasi dari solusi secara acak dan melakukan pencarian solusi optimum (*optima*) dengan melakukan pembaruan pada generasi selanjutnya secara iteratif. Namun dalam faktor kompleksitas pengimplementasian, PSO dianggap sangat *powerful* karena dengan hanya membutuhkan sedikit parameter – lebih sedikit dari GA – dan tidak diperlukannya *evolution operator* seperti *crossover* dan *mutation*, PSO mampu memberikan hasil yang konsisten dan memuaskan seperti GA. PSO tidak menjamin ditemukannya *optima* yang diinginkan, namun PSO memiliki kinerja yang baik pada problem dengan domain yang memiliki dimensi tinggi, *non-convex*, dan *non-continuous* atau diskrit.

Pada laporan ini akan dijelaskan bagaimana mengimplementasikan PSO sederhana untuk menyelesaikan kasus matematis memaksimalkan fungsi trigonometri dua variabel.

## Problem

Seperti pada laporan sebelumnya (IF184981\_TK\_05111340000107\_Son-Ardhynata.pdf), penulis akan menggunakan problem yang sama yakni memaksimalkan fungsi trigonometri:

Degan presisi 4 angka di belakang koma.

## Ide

Pengimplementasian PSO dalam laporan ini akan menggunakan pendekatan *object-oriented programming*. Pendekatan OOP dianggap sangat tepat dalam pengimplementasian PSO karena sifat dari PSO yang menirukan objek dunia nyata seperti kumpulan burung atau sekelompok ikan. Objek-objek tersebut memiliki karakteristik dan *behavior* yang dapat diformulasikan menjadi *class* dalam *object-oriented programming*.

## Spesifikasi Perangkat

#### Software:

Programming language : Python 3

IDE : Jupyter Notebook

Library : numpy (Python)

#### Hardware:

Processor : Intel(R) Core(TM) i7-7700HQ

RAM : 16 GB

## Link Code

Code pada laporan ini dapat di-download/pull pada alamat <https://github.com/Ardhynata/ParticleSwarmOptimization.git> dan dapat di-run langsung menggunakan Jupyter Notebook.

## Metode Penyelesaian

#### 1. Mempersiapkan Library dan Class

Library yang digunakan dalam implementasi PSO ini adalah *numpy*, sebuah library paling umum digunakan untuk menyelesaikan kasus pemrograman yang berhubungan dengan permasalahan matematika. Numpy akan digunakan untuk keperluan sebagai berikut:

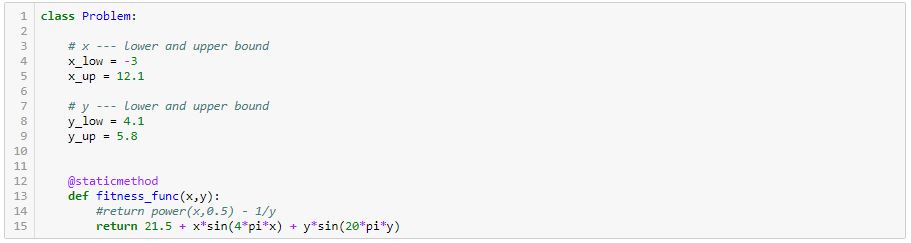
* Mencari nilai sinus dari persamaan yang disebutkan di atas
* Melakukan *randomization* bilangan *float*
* Menggunakan nilai π dengan presisi hingga 16 angka di belakang koma



Gambar 1. Libary Numpy

Class yang akan digunakan adalah sebagai berikut:

* Class Problem untuk menampung domain dari problem dari persamaan yang dijelaskan di atas dan *fitness function* yang dalam hal ini adalah persamaan itu sendiri.



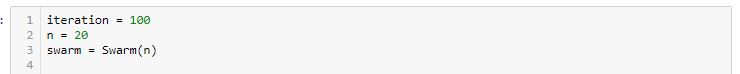
Gambar 2. Class Problem

* Class Particle yang merupakan representasi dari solusi-solusi yang mungkin dari persamaan di atas. Class ini bertanggungjawab dalam menghitung *fitness* sebuah partikel yang direpresentasikan oleh sebuah koordinat (x,y) dan menghitung kecepatan (*velocity*) dari partikel tersebut yang direpresentasikan berupa vektor. Partikel-partikel ini memiliki juga memiliki atribut bernama *pBest* untuk menyimpan nilai *fitness* terbaik dari sebuah individu partikel yang nantinya digunakan untuk mencari *gBest* (dijelaskan pada class Swarm) yang berpengaruh pada pergerakan individu menujusolusi yang dicari.
* Class Swarm memiliki *class attribute* yakni *gBest* yang berguna untuk menyimpan nilai *fitness* terbaik yang pernah dicapai oleh populasi. Nilai *gBest* pada iterasi terakhir adalah solusi dari persamaan di atas.

#### 2. Alur Penyelesaian

##### i. Initial Population

Metode PSO dimulai dengan men-*generate* populasi awal dengan menentukan jumlah individu partikel yang akan digunakan untuk mencari solusi. Proses ini dilakukan dengan melakukan *randomization* untuk men-*generate* koordinat dalam *vector space* sebanyak jumlah yang ditentukan.



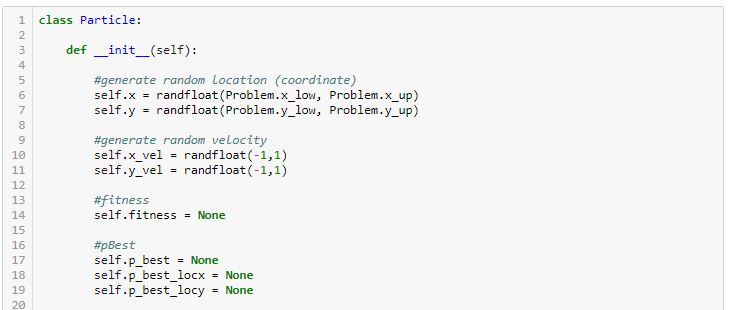
Gambar 3. Instantiation Objek Swarm

Proses *instantiation* di atas akan menginisialisasi variable *n* (jumlah partikel) dalam class *Swarm*



Gambar 4. Inisialisasi attribute objek Swarm

Proses di atas akan men-generate partikel-partikel sejumlah *n* dari class *Particle* berikut:



Gambar 5. Inisialisasi variabel-variabel Particle

##### ii. Evaluation

Proses evaluasi adalah proses menghitung *fitness* dari sebuah partikel. Proses ini dilakukan dengan memanggil *method calc\_fitness* pada *class Swarm*.



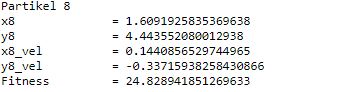
Gambar 6. Method calc\_fitness pada class Swarm

*Method* di atas akan memanggil *method calc\_fitness* pada seluruh objek-objek Particle.



Gambar 7. Method calc\_fitness pada Particle

Sampai pada langkah ini, kita sudah bisa mengetahui nilai *fitness* semua partikel.

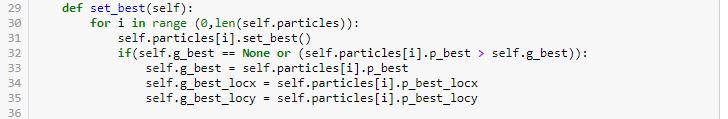


Gambar 8. contoh output Fitness pada Partikel 8 dari 20 partikel

##### iii. Menentukan *pBest* dan *gBest*

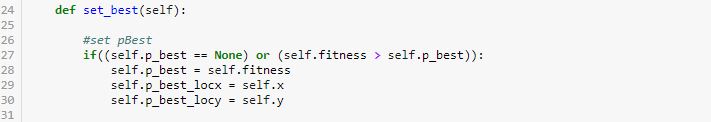
Langkah selanjutnya adalah menentukan *local best* (*pBest*) dan *global best* (*gBest*). Pengertian dari *local best* adalah nilai *fitness* terbaik yang pernah dialamai oleh sebuah partikel, sedangkan *global best* adalah nilai *fitness* terbaik yang pernah dialami oleh populasi.

Untuk menentukan nilai *pBest* dan *gBest*, kita harus memanggil *method set\_best* pada *class Swarm*.



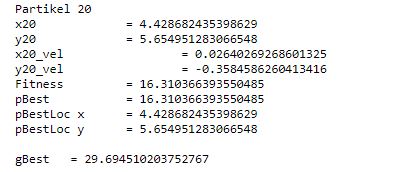
Gambar 9. Method set\_best pada Swarm

*Method* di atas akan memanggil *method set\_best* pada semua objek-objek Particle dan juga memberikan nilai pada *class attribute g\_best, g\_best\_locx,* dan *g\_best\_locy*.



Gambar 10. Method set\_best pada Particle

Langkah-langkah di atas akan menghasilkan output sebagai berikut:



Gambar 11. Contoh Output pada partikel 20 dan nilai gBest iterasi pertama

##### iv. Menghitung *velocity* baru

Sebelum menuju ke iterasi selanjutnya terlebih dahulu kita harus meng-*update* lokasi dari partikel-partikel yang ada. Perpindahan partikel-partikel tersebut dipengaruhi oleh nilai *velocity* yang dimiliki partikel dan juga lokasi dari *pBest* dan *gBest.*

Penghitungan *velocity* baru mengikuti rumus berikut ini:

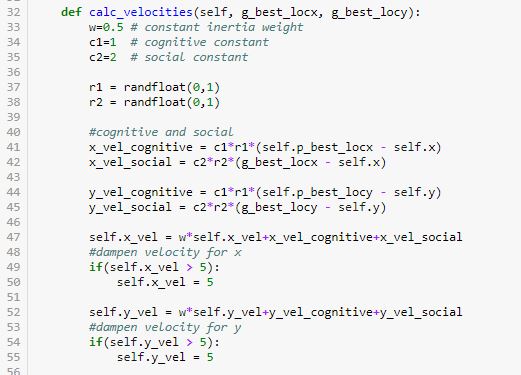


Implementasi untuk menghitung *velocity* baru adalah dengan memanggil *method* ***calc\_velocities*** pada *class* ***Swarm***



Gambar 12. Method calc\_velocities pada Swarm

*Method* di atas akan memanggil *method calc\_velocities* pada semua objek-objek Particle



Gambar 13. . Method calc\_velocities pada Particle

Pada *method* di atas telah ditambahkan *if-function* untuk membatasi penambahan kecepatan yang terlalu besar (*velocity dampening*). Sehingga untuk setiap *velocity* baru yang melebihi *threshold* yang ditentukan (dalam hal ini di-set pada angka 5) akan di-set tepat pada nilai *threshold*.

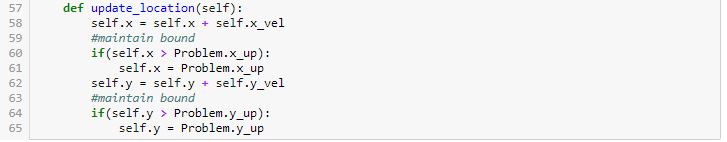
##### v. Meng-*update* lokasi partikel-partikel

Langkah selanjutnya adalah meng-update lokasi dari partikel-partikel yang ada dalam *vector space.*

Untuk melakukan *update* lokasi dilakukan dengan memanggil *method update\_location* pada *class Swarm*



Gambar 14. Method update\_location pada Swarm

*Method* di atas akan memanggil *method update\_location* pada setiap objek-objek *Particle*

Gambar 15. Method update\_location pada Particle

*Method* di atas ditambahkan *if-function* untuk membatasi agar partikel tidak bergerak keluar batas yang ditentukan.

Lakukan secara berulang-ulang ke lima langkah di atas hingga beberapa kali perulangan (dalam implementasi ini diulang hingga 100 kali perulangan).

## Hasil Implementasi

Setelah dilakukan perulangan hingga 100 kali didapatkan *gBest ­*= 38.73280596954382. Hasil ini adalah hasil yang sangat tepat hingga 4 angka di belakang koma.

Hasil ini dapat di-*benchmark* di WolframAlpha di alamat [https://www.wolframalpha.com/input/?i=maximize+21.5+%2B+x\*sin(4\*pi\*x)+%2B+y\*sin(20\*pi\*y)+with+-3.0+%3C%3D+x+%3C%3D+12.1+and+4.1+%3C%3D+y+%3C%3D+5.8](https://www.wolframalpha.com/input/?i=maximize+21.5+%2B+x*sin(4*pi*x)+%2B+y*sin(20*pi*y)+with+-3.0+%3C%3D+x+%3C%3D+12.1+and+4.1+%3C%3D+y+%3C%3D+5.8)

## Kesimpulan

Dari percobaan di atas dapat dibuktikan bahwa Particel Swarm Optimization (PSO) mampu memberikan solusi yang tepat hingga 4 angka di belakang koma. Hal-hal yang mempengaruhi performa PSO antara lain adalah jumlah populasi awal dan nilai *velocity dampening*. Semakin banyak partikel yang digunakan, kemungkinan besar akan memberikan hasil yang baik dan menemukan angka yang tepat sebagai *threshold* untuk *velocity dampening* juga akan memberikan hasil yang baik, threshold ini harus diatur sedemikian rupa sehingga tidak terlalu membatasi pergerakan partikel namun juga tidak menjadikan partikel bergerak terlalu bebas.