# Sujets TP d’application

# Par groupe de 3 : 9 groupes.

**TP simples :**

- Faire un algo mémétique le comparer à l'algo simple, tester sur des Weierstrass et sur Rastrigin.
- Programmer un niching et tester sur Rastrigin en grande dimension.
- Comparaison de différents outils de régression symbolique.
- Comparaison de différents algorithmes d'optimisation multi-objectifs.
- Programmer un algorithme multi-population (modèle en ilôts), le comparer à un algo classique.

**TP de difficulté moyenne :**

- Programmer et tester différentes mesures de diversité.
- Labyrinthe avec ArenaBot.
- Régression symbolique multi-objectif.
- Optimisation d'un réseau de neurones.

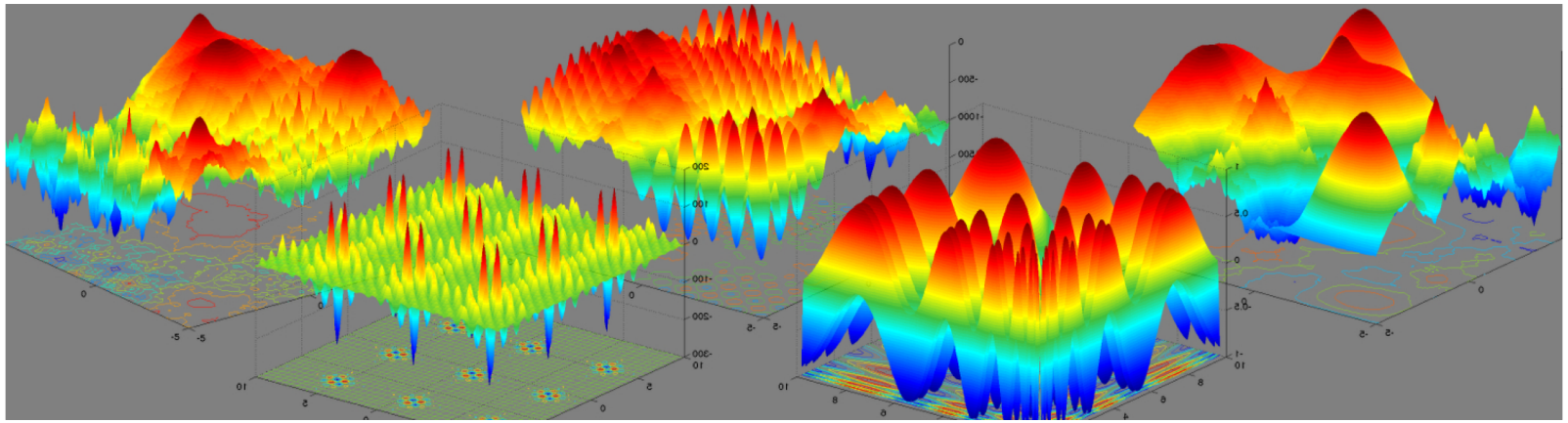
**TP plus compliqués :**

- Co-evolution coopérative : problème « Lamps ».
- Co-évolution compétitivie : modèle prédateur/proie.
- MicroGP4: opérateurs pour le Linear Genetic Programming.
- Evolution interactive d'ensembles de Julia.

**Sujet « sur mesure » sur demande**

# Niching : Identification des différents optima

- **Fonctions multi-modales :**



- **L'optimisation évolutionnaire classique converge vers  un seul optimum local.**
- **Solution : Forcer la population à converger vers plusieurs optima plutôt qu'un seul.**
- **Implémenter à l'aide d'inspyred une ou plusieurs méthodes de niching.**
- **Tester les méthodes sur des fonctions multi-modales et comparer leurs performances avec l'optimisation évolutionnaire classique (code TP1).**

*Niveau de difficulté* ★

# Algorithme évolutionnaire mémétique

- **Optimisation évolutionnaire classique, mais dotée de capacités “d’apprentissage”. En général : rajout d’une optimisation locale lors de l’évaluation de chaque individu.**

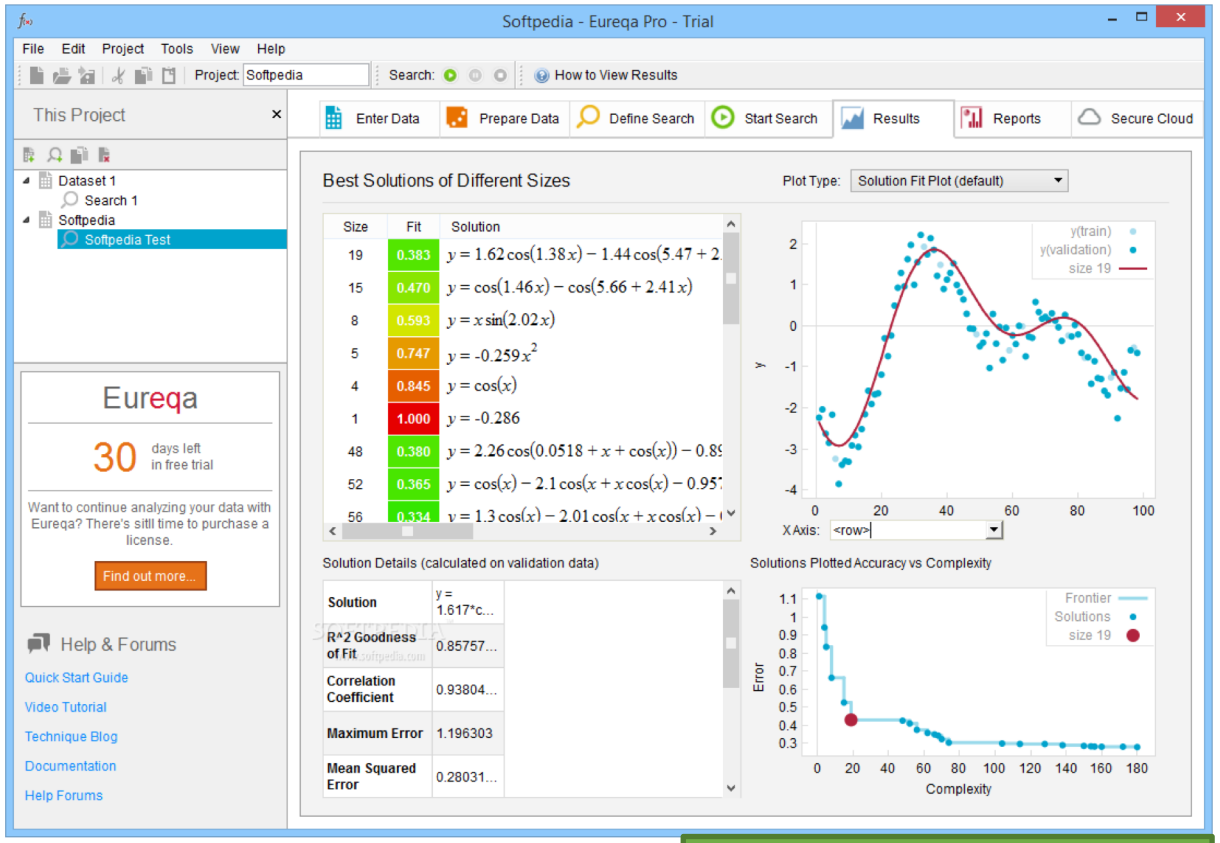
- **Implémentation avec inspyred.**

- **Paramétrage et comparaison de l’approche mémétique avec l’approche classique.**

***Niveau de difficulté***

# Comparaison de différents outils de régression symbolique

- **Comparaison entre Eureqa, GSGP (en C++), et GPlearn (en Python) :**
  - **Temps de calcul**
  - **Qualité des solutions proposées**
  - **A tester sur différents jeux de test.**

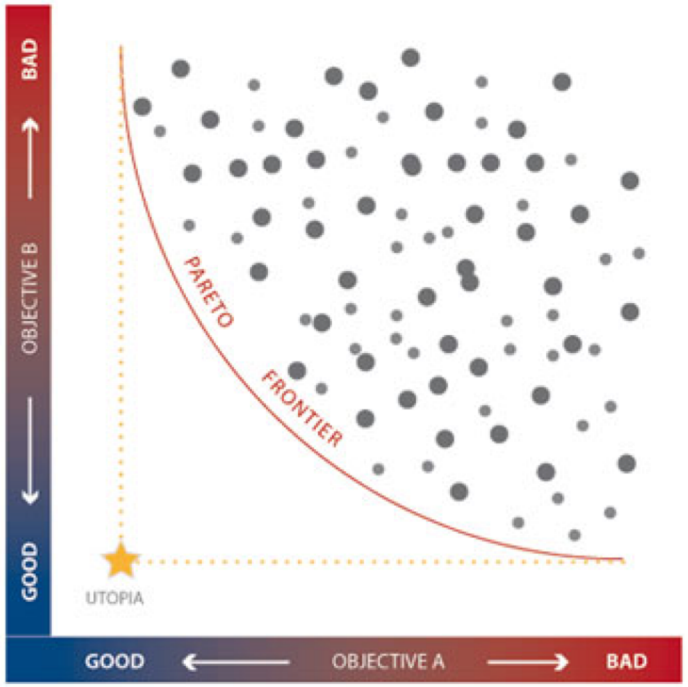



***Niveau de difficulté*** ★

# Comparaison de différents algorithmes d'optimisation multi-objectifs

- **Comparer les performances entre NSGA-II, NSGA-III et MOEA/D**
  - **Temps de calcul,**
  - **Qualité des fronts obtenus (programmer différentes métriques).**

- **Utilisation de différents problèmes multi-objectifs, avec un nombre d'objectifs variable.**




*Niveau de difficulté* ★

# Algorithme multi-populations : modèle à îlots

- Point de départ : https://pythonhosted.org/inspyred/examples.html#island-models
- Comparer avec un modèle mono-population utilisant le même nombre d'évaluations.
- Faire varier le nombre d'îlots, les fréquences de communication.
- Sur différentes fonctions-test (Rastrigin, Weierstrass).

Niveau de difficulté ★

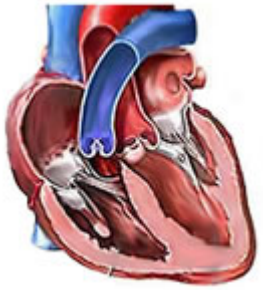
# Mesurer et suivre la diversité

- Programmer une mesure de diversité de population.
  - Mesures géométriques : somme, max, moyenne des distances entre toutes les paires possibles d'individus (définir une distance sur l'espace de recherche : phénotypique, génotypique, fitness ?)
  - Mesures statistiques (variance, écart-type, caractéristiques calculées sur les histogrammes des valeurs de gènes, entropie de Shannon)
- Affichage à chaque génération (petite interface de visualisation).
- Tester et comparer les mesures de diversité (différents paramétrages d'AE sur quelques fonctions-test).

Niveau de difficulté ☆☆

# Optimisation de réseaux de neurones

- **Prérequis : une bonne connaissance en apprentissage de réseaux de neurones artificiel.**
- **Jeu de donnée : Diagnostic de maladies cardiaques.**
- **Apprendre les poids et la structure d'un réseau de neurones par optimisation évolutionnaire.**
- **Réduire au maximum la complexité du réseau tout en gardant un niveau de prédiction satisfaisant (multi-objectifs ?).**



*Niveau de difficulté* ★★

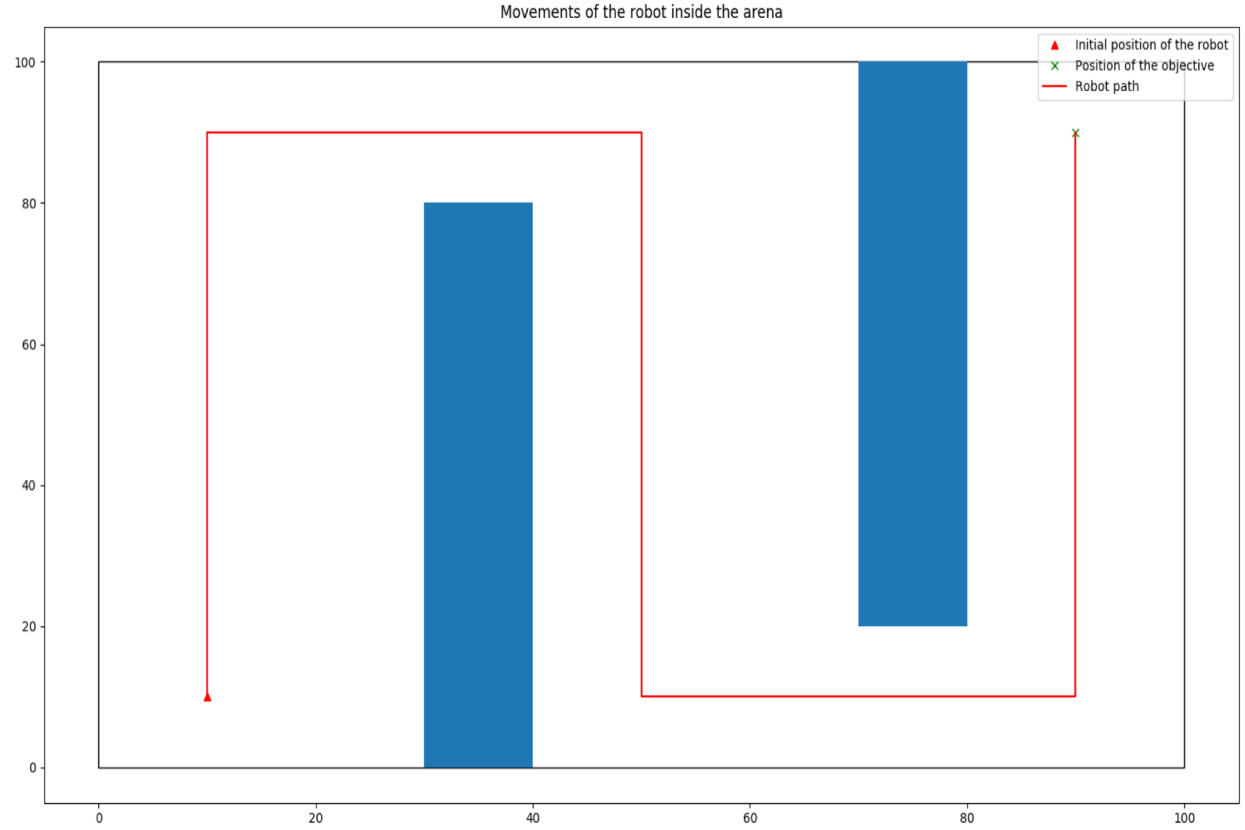
# Régression symbolique multi-objective

- [**plus cool**] Utiliser le code de gplearn (http://github.com/trevorstephens/gplearn) et inspyred pour créer une version multi-objective (erreur, complexité) de la régression symbolique, “à la Eureqa”, mais implémenté en Python. Comparer avec Eureqa sur le nombre d’évaluation.
- [**plus facile**] Régler le paramètre **parsimony_coefficient** de la fonction **SymbolicRegressor** de gplearn pour obtenir un front Pareto (erreur, complexité). Comparer avec Eureqa sur le nombre d’évaluations.

*Niveau de difficulté* ☆☆

# Labyrinthe

- A l’aide d’inspyred, écrire un algo GP pour aider un “robot” à trouver son chemin dans un labyrinthe




- Le robot va recevoir une liste de commandes (“move 80”, “rotate 90”, “move 20”, …) et la fitness évalue la distance de l'objectif.

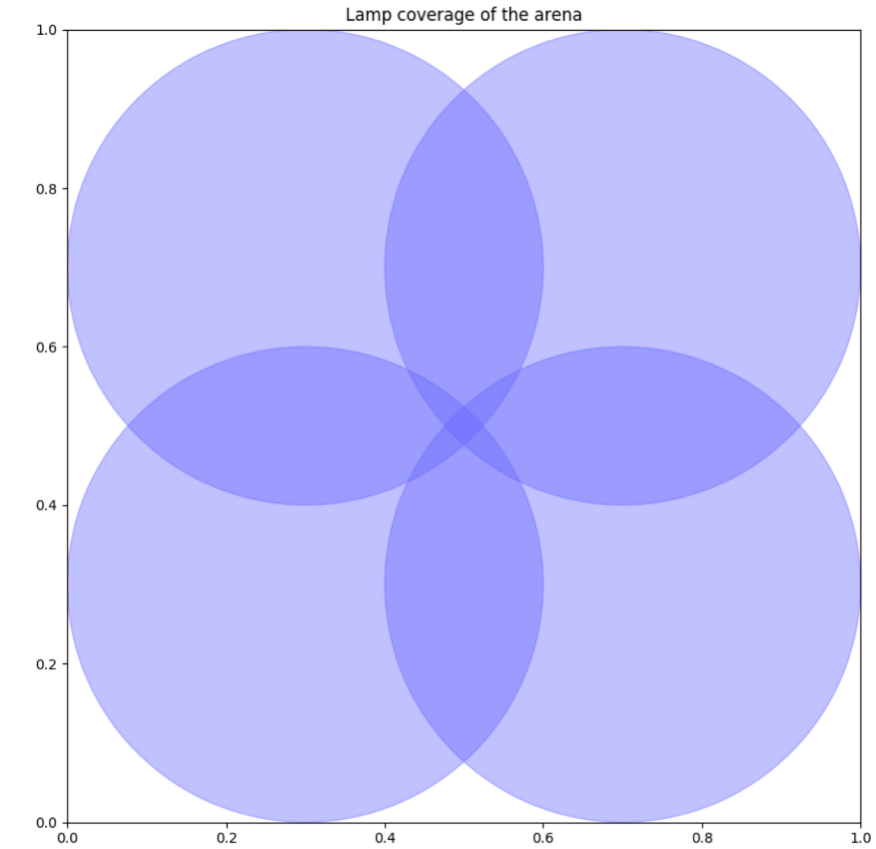
  Le robot ne peut pas passer à travers les murs.

  La fonction dans **arenabot.py** calcule la fitness pour une liste de commandes donnée.

Niveau de difficulté ★★

# Co-evolution coopérative

- Avec un cas d'étude "lamps", utiliser inspyred (ou autre) pour créer un "cooperative co-evolutionary algorithm"




- Chaque individu est un lampe (un cercle), et l'objectif global c'est de couvrir le carré  avec le minimum de lampes. La fitness de chaque lampe est son recouvrement du carré.

***Niveau de difficulté*** ★★★

# Co-évolution compétitive : prédateur/proie

- Point de départ = modèle à base d'agents wolf/sheep : https://github.com/projectmesa/mesa/tree/master/examples/wolf_sheep
- Deux populations : les loups et les moutons.
- Génomes = caractéristiques individuelles des agents (loups ou moutons).
- Evaluation = chaque individu a un niveau d'énergie qui correspond à sa fitness.
- Sélection / Croisement / mutation  évolutionnaire pour chaque population (à programmer !).

# MicroGP4: opérateurs pour Linear Genetic Programming

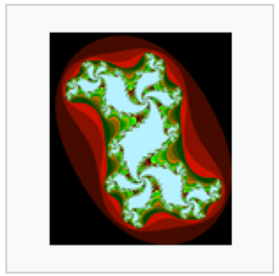
- MicroGP 4.0 est un algo de LGP capable de lire un fichier en JSON avec la description de stringes à générer, et suivre les instructions
- Au moment, il n'a pas d'opérateurs (il est un généreateur aléatoire de stringes).
- Implémenter des opérateurs (mutations, cross-overs?) pour MicroGP pour un cas d'étude, et tester l'efficacité de l'algo avec et sans opérateurs.

Niveau de difficulté ★★★

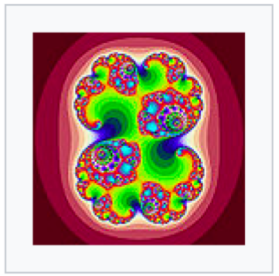
# Exploration interactive d'images fractales : ensembles de Julia.

- **Ensembles de Julia :**

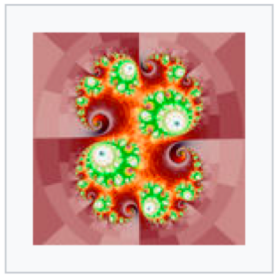
  Comportement de la suite $z_{n+1} = z_n^2 + c$ dans le plan complexe.



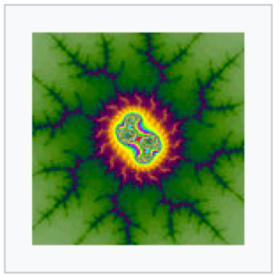
Ensemble de Julia de $z^2 + c$ avec $c = 0.3 + 0.5i$.



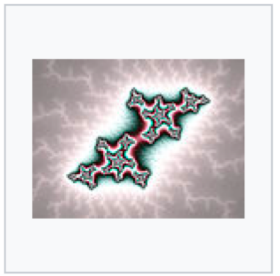
Ensemble de Julia pour $c = 0.285 + 0.01i$ (courbes de niveau du nombre d'itérations).



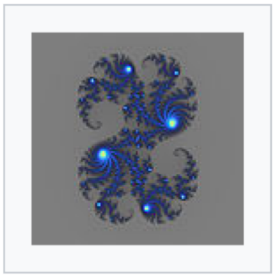
Le même ensemble avec une coloration différente (nombre d'itérations et pavage selon le secteur d'arrivée).



Détail de l'ensemble de Julia pour $c = -1.417022285618$.



Détail de l'ensemble de Julia pour $c = -0.038088 + 0.975$.



Ensemble de Julia pour $c = 0.285 + 0.013i$.

- **Génome : c = x + iy + table de couleurs + ?**
- **Evaluation interactive par un slider.**
- **Présentation d'une population, vote et itération suivante.**

***Niveau de difficulté*** ★★★★