Ingénierie des systèmes orientés-objet

Laboratoire 1

Résolution de problèmes d'ingénierie par évolution différentielle

Table des matières

Introduction

Ce premier laboratoire consiste à réaliser un programme informatique pouvant trouver la solution à trois problèmes d'ingénierie de natures différentes.

La résolution de ces problèmes est réalisée par l'utilisation de l'algorithme à évolution différentielle. Ainsi, le même outil permettra l'identification de solutions viables pour les trois problématiques.

Afin d'assurer une bonne mise en place des notions reliées au paradigme de la programmation orientée objet, vous serez fortement guidé vers l'implémentation à réaliser.

Objectifs

Les objectifs pédagogiques de ce laboratoire sont énumérés en ordre décroissant d'importance:

- Prendre contact avec le paradigme de la programmation orienté objet.
- Apprendre et mettre en pratique la syntaxe et la sémantique du langage C++.
- Autant conceptuellement que concrètement à l'aide du langage C++, comprendre :
    - la notion d'encapsulation (regroupement et intégrité)
    - la différence entre une classe et un objet
- Apprécier les capacités de l'algorithme à évolution différentielle.
- Être capable de réaliser un programme dont le schéma de conception est existant quoiqu'incomplet.
- Se familiariser avec l'environnement de développement Microsoft Visual Studio autant pour la rédaction de code que pour le débogage.
- Mettre en application une norme de codage et les bonnes pratiques associées.

Résumé du projet

On vous demande de créer un logiciel capable de résoudre trois problèmes d'ingénierie :

1. Problème de la boîte ouverte
2. Identifier les deux optima de la fonction « *peaks* » de Matlab
3. Problème d'optimisation de la production dans une entreprise manufacturière

La réalisation de l'application aborde implicitement trois aspects distincts importants du développement logiciel:

a. L'implémentation de l'algorithme de l'évolution différentielle (l'engin de résolution)
b. La modélisation et l'implémentation des solutions aux trois problèmes donnés
c. La modélisation et l'implémentation d'une interface utilisateur (en mode console)

Le développement de ce projet consiste à utiliser le paradigme orienté objet et de développer son sens critique sur la notion de l'encapsulation, autant sur les concepts que de la programmation en C++. On retrouve les concepts de l'encapsulation ainsi :

a. Évolution différentielle : Implémentation d'une conception imposée
b. Les 3 problèmes : Utilisation d'une encapsulation existante
c. L'interface utilisateur : Conception et implémentation d'une encapsulation

Évolution différentielle

Introduction

L'évolution différentielle (ED) est un algorithme[1] de résolution de problème issue de la famille des algorithmes évolutionnaires[2]. Il peut être classé parmi les méthodes métaheuristiques[3,4] stochastiques[5] d'optimisation[6]. Plus précisément, il correspond à une variante des algorithmes génétiques[7] jointe à une méthode de recherche vectorielle[8].

Consulter ces articles pour plus de détails :

- métaheuristique
- algorithme évolutionniste
- algorithme génétique
- évolution différentielle (en anglais)

L'ED est un outil souvent utilisé dans divers domaines du génie à titre d'algorithme d'intelligence artificielle. Malgré le fait qu'il soit simple à implémenter, il donne généralement d'excellents résultats selon la nature du problème et de la modélisation faite.

On présente ici une version minimaliste de l'évolution différentielle. Si vous désirez plus d'informations sur les nuances et possibilités, discuter du sujet avec l'enseignant et le chargé de laboratoire.

Espace de solution

On représente le problème à optimiser par la fonction *g* exprimée dans un espace à *D* dimensions dans les réels. Cet espace est nommé espace de solution Θ.

$$g: \mathbb{R}^D \tag{1}$$

g est nommée la fonction objective (« goal function » en anglais). C'est elle qui représente le problème à optimiser. Le problème se pose dans un espace à *D* dimensions où chacune représente une variable du système.

Ainsi, une solution est représentée par un vecteur de taille *D* où chaque scalaire correspond à un paramètre. Ici *s* est un vecteur de taille *D*.

$$\vec{s} \in \mathbb{R}^D \tag{2}$$

[1] Algorithme : Séquence structurée et ordonnée d'instructions pour résoudre un problème spécifique.
[2] Algorithme évolutionnaire : Méthode d'optimisation inspirée des processus naturels de sélection et d'évolution pour trouver des solutions à des problèmes complexes.
[3] Heuristique : Approche simplifiée et intuitive permettant la résolution de problèmes difficiles en utilisant des solutions approximatives plutôt qu'exactes.
[4] Métaheuristique : Ensemble de techniques d'optimisation de haut niveau qui guident les heuristiques pour explorer l'espace de recherche de manière efficace.
[5] Stochastique : Processus ou modèle impliquant des variables aléatoires et des probabilités pour décrire des phénomènes incertains.
[6] Optimisation : Processus visant à trouver la meilleure solution ou les meilleures valeurs pour un problème donné en maximisant ou minimisant une fonction objective.
[7] Algorithme génétique : Type d'algorithme évolutionnaire inspiré de la génétique pour rechercher des solutions optimales en utilisant des opérations de sélection, de croisement et de mutation.
[8] Méthode de recherche vectorielle : Approche d'optimisation qui utilise des vecteurs pour représenter et manipuler des solutions potentielles dans un espace multidimensionnel.

Même si l'ED tend vers une meilleure solution, les algorithmes évolutifs ne peuvent garantir l'identification de la meilleure solution : c'est-à-dire l'extremum global. Souvent, un extremum local est identifié sans certitude ou connaissance sur l'existence ou non de l'extremum global. L'extremum se pose ainsi :

$$\forall \vec{b} \in \mathbb{R}^D : g(\vec{a}) \leq g(\vec{b}) \tag{3}$$

$$\forall \vec{b} \in \mathbb{R}^D : g(\vec{a}) \geq g(\vec{b}) \tag{4}$$

L'équation (3) est utilisée pour des problèmes de minimisation et l'équation (4) pour des problèmes de maximisation. Dans les deux cas, $\vec{a}$ représente la meilleure solution identifiée.

Afin de simplifier et standardiser le développement, on utilise souvent une seule implémentation parmi (3) et (4). Dans ce cas, on passe par une autre fonction *f* afin de modifier et adapter le résultat de *f*. Par exemple :

$$\forall \vec{b} \in \mathbb{R}^D : f(g(\vec{a})) \geq f\left(g(\vec{b})\right) \tag{5}$$

La fonction d'évaluation (ou d'adaptation ou encore « fitness function » en anglais) est souvent très simple :

$$f(\vec{x}) = \vec{x} \tag{6}$$

$$f(\vec{x}) = -\vec{x} \tag{7}$$

$$f(\vec{x}) = \begin{cases} \frac{1}{\vec{x}} & si\ \vec{x} \neq 0 \\ 0 & sinon \end{cases} \tag{8}$$

$$f(\vec{x}) = a \cdot \vec{x} + b \tag{9}$$

Les fonctions (6) à (9) sont des exemples simples de *f* : identité (6), négation (7), inverse (8) et transformation linéaire (9). Par exemple, on utilise généralement (6) lorsque *g* exprime naturellement le fait que les solutions favorables ont de plus hautes valeurs.

Notation utilisée

- $\mathbb{R}^D$ espace des réels à *D* dimensions
- Θ espace de solution $\Theta \equiv \mathbb{R}^D$
- D le nombre de dimensions (nombre de paramètres du problème)
- P le nombre de solutions (population)
- G le nombre de générations
- G_{max} le nombre de générations maximum
- $\vec{x}$ la population, un vecteur de taille *P* contenant toutes les solutions en cours
- $\vec{m}$ les mutants, un vecteur de taille *P* contenant toutes les solutions mutantes
- $\vec{t}$ les tests, un vecteur de taille *P* contenant toutes les solutions tests (« *trial* »)
- $\vec{s}$ une solution, un vecteur de taille *D*
- $\overrightarrow{s_{p,g}}$ un vecteur de dimension *D* représentant la p^e solution à la g^e génération
- $v_{i,p,g}$ un scalaire de *v* représentant le i^e paramètre de la p^e solution à la g^e génération
- $\vec{l}$ un vecteur de dimension *D* représentant les bornes inférieures de $\vec{s}$
- $\vec{u}$ un vecteur de dimension *D* représentant les bornes supérieures de $\vec{s}$
- F le poids différentiel (un paramètre de l'évolution différentielle)
- CR la probabilité de croisement (« *crossover rate* ») (un paramètre de l'ED)
- g la fonction objective (« *objective function* »)

- f la fonction d'évaluation (« *fitness function* »)
- $rand(l,u)$ une fonction générant un nombre aléatoire borné : $rand(l,u) \in [l,u]$
- $I(\dots)$ la fonction d'initialisation
- $M(\dots)$ la fonction de mutation
- $C(\dots)$ la fonction de croisement
- $S(\dots)$ la fonction de sélection (ou de remplacement)

Algorithme

L'algorithme est découpé en six opérations logiques distinctes.

Identification du problème

Les paramètres suivants sont fixés au départ de la simulation et représentent le contexte général de résolution:

$$D, P, G_{max}, F, CR, \vec{l}, \vec{u}, f \text{ et } g.$$

Finalement, les constantes sont limitées à :

$$0 \leq F \leq 2$$

$$0 \leq CR \leq 1$$

Création de la population

Cette tâche consiste à produire le vecteur $\vec{x}$ de taille *P*. À ce moment, les *P* solutions sont non initialisées.

Initialisation

$$x_{i,p,0} = I(l_i, u_i) = rand(l_i, u_i) \quad (10)$$
$$\forall i, i = 1, \dots, D$$
$$\forall p, p = 1, \dots, P$$

```
// x est la population en cours
pour tous les solutions de x (p)
     pour toutes les valeurs de xp (i)
          xpi = rand(li, ui)
```

Mutation

$$m_{i,p} = M\left(\overrightarrow{s1}, \overrightarrow{s2}, \overrightarrow{s3}\right) = s1_{i,p} + F \cdot \left(s2_{i,p} - s3_{i,p}\right) \quad (11)$$

$$\forall i, i = 1, \dots, D$$

$$\forall p, p = 1, \dots, P$$

Où $\overrightarrow{s1}$, $\overrightarrow{s2}$ et $\overrightarrow{s3}$ sont trois solutions de $\vec{x}$ sélectionnées aléatoirement pour chaque *p*. Ils doivent être différents les uns des autres ainsi que de $\overrightarrow{s_p}$.

```
// x est la population en cours
// m est la population mutante
pour toutes les solutions de m (p)
     // sélectionner 3 solutions distinctes
     s1, s2, s3 = select_3_solution_from_x_and_distinct_of_p(...)
     pour toutes les valeurs de mp (i)
          mpi = si + F * (s2i - s3i)
```

Croisement

$$\overrightarrow{t_{i,p}} = C\left(\overrightarrow{x_{ip}}, \overrightarrow{m_{ip}}, R\right) = \begin{cases} m_{i,p} & si\ (i = R\ \cup\ rand(0,1) < CR) \\ x_{i,p} & sinon \end{cases} \quad (12)$$

$$\forall i, i = 1, \dots, D$$

$$\forall p, p = 1, \dots, P$$

Où *R* est un nombre aléatoire générée pour chaque solution de la population. $R = rand(1, D)$.

```
// x est la population en cours
// m est la population mutante
// t est la population test
pour toutes les solutions de t (p)
     R = rand(1, D)
     pour toutes les valeurs de tp (i)
          tpi   = mpi      si (i = R) ou rand(0, 1) < CR
                = xpi      sinon
```

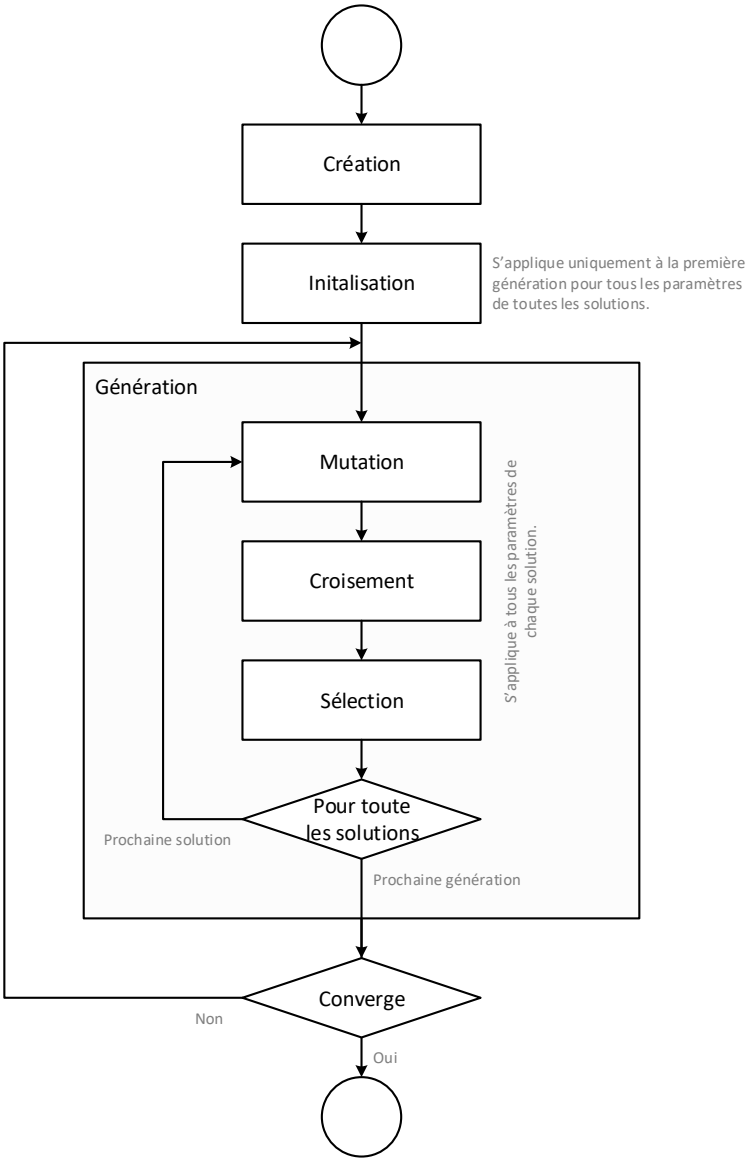
Sélection

$$\overrightarrow{x_{p,g+1}} = C\left(\overrightarrow{x_{p,g}}, \overrightarrow{t_{p,g}}\right) = \begin{cases} \overrightarrow{t_{p,g}} & si\ f\left(g\left(\overrightarrow{t_{p,g}}\right)\right) \geq f\left(g\left(\overrightarrow{x_{p,g}}\right)\right) \\ \overrightarrow{x_{p,g}} & sinon \end{cases} \quad (13)$$

Dans le cas où on réalise une maximisation.

```
// x est la population en cours
// x' est la prochaine population
// t est la population test
pour toutes les solutions de x (p)
     x'p  = tp       si f(g(tp)) >= f(g(xp))
          = xp       sinon
```

Déroulement de l'algorithme



Instructions du laboratoire

Présentation générale

Pour ce projet, vous devez créer un logiciel solutionnant trois problèmes d'ingénierie différents. La résolution de chaque problématique doit être réalisée par le même engin de résolution, soit l'algorithme de l'évolution différentielle.

L'idée est de créer quelques classes génériques réutilisables pour certains aspects du projet et d'autres plus spécifiques pour les particularités spécifiques de l'application. Il est important de découper le travail en plusieurs sous-classes favorisant la réutilisabilité de votre travail.

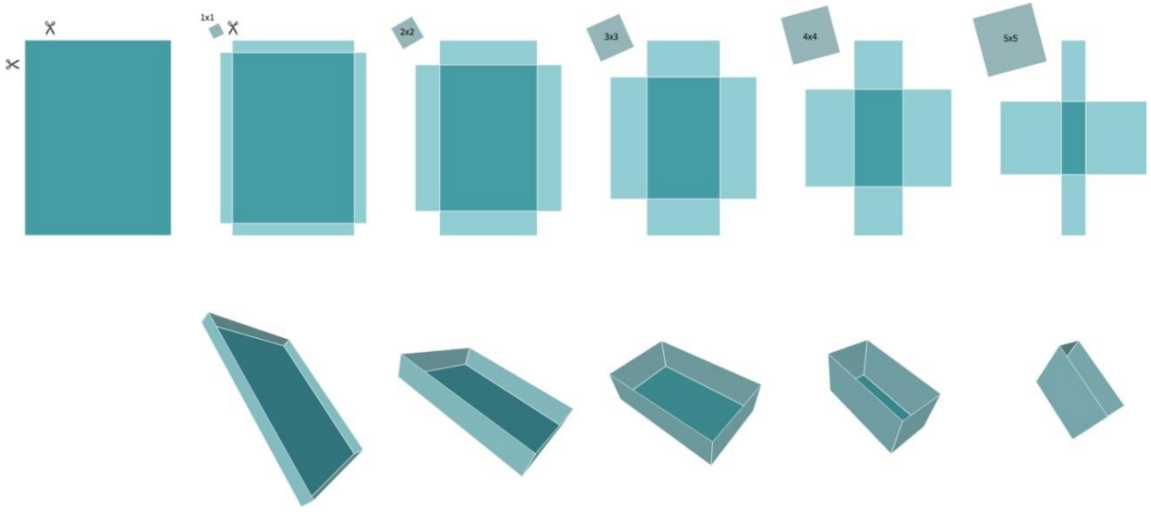
On vous donne une structure plutôt stricte à respecter et pour laquelle il reste tout de même plusieurs sous éléments et détails à concevoir et à réaliser.

Problématiques à résoudre

Problème de la boîte ouverte

Le problème d'optimisation de la boîte ouverte est un peu le « *hello world* » du domaine de l'optimisation. C'est-à-dire que ce problème permet d'aborder un exemple simple d'optimisation. Le problème est suffisamment simple pour mettre l'emphase sur les considérations algorithmes autour du problème. C'est une excellente opportunité d'aborder le monde de l'optimisation.

Le problème se pose ainsi, considérez une feuille de carton rectangulaire de dimension *w* x *h*. On désire découper une zone carrée de longueur *x* dans chacun des quatre coins de façon à ce que lorsque les quatre zones latérales sont repliées, la feuille de carton forme une boîte avec une ouverture sur le dessus.



On désire déterminer la taille du carré à découper (*x*) qui maximise le volume de la boîte ouverte obtenue.

Pour la résolution de ce problème, on suppose les dimension *w* et *h* connues et fixes :

- la largeur de la feuille de carton : *w* = 100
- la hauteur de la feuille de carton : *h* = 50

Toutefois, le programme doit poser les questions suivantes :

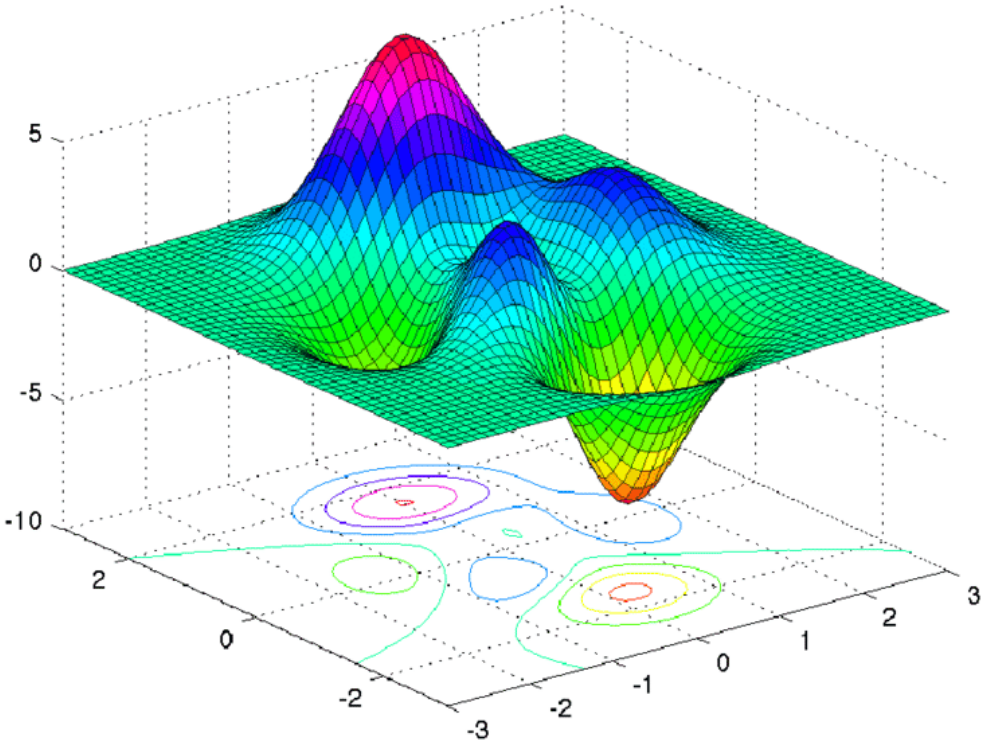
- Quelle est la valeur du paramètre *F,* le poids différentiel :
  - par défaut = 2
  - la valeur saisie doit être bornée à l'intervalle [0.0, 2.0]
- Quelle est la valeur du paramètre *CR*, la probabilité de croisement :
  - par défaut = 1
  - la valeur saisie doit être bornée à l'intervalle [0.0, 1.0]

Optima de la fonction « *peak* »

Vous devez trouver le minimum et le maximum de la célèbre courbe `peaks` de *Matlab* pour les intervalles donnés. Vous ne devez utiliser qu'une simulation pour trouver les deux coordonnées.

$$peaks(x,y) = 3(1-x)^2 e^{-(x^2+(y+1)^2)} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-(x^2+y^2)} - \frac{1}{3} e^{-((x+1)^2+y^2)}$$

$$x, y \in [-3,3]$$



Structure de pointage :

- deux simulations pour trouver le minimum et le maximum : 75%
- une simulation pour trouver simultanément le minimum et le maximum : 100%
- bonus : trouver les trois maxima locaux ainsi que les deux minima locaux avec une seule simulation : 125% (attention, plus difficile qu'il n'y parait, s'y attarder seulement à la fin du projet).

Pour la résolution de ce problème, le programme doit poser la question suivante :

- Quelle est la taille de la population *P* (défaut = 10) ?
- Quel est le nombre de générations G_{max} (défaut = 100) ?
- Quelle est la précision des points flottants à afficher (défaut = 6) ?

Optimisation de la production pour une entreprise manufacturière

Une entreprise manufacturière produit n types de produits différents et désire maximiser son profit tout en tenant compte des contraintes liées aux ressources disponibles. Pour simplifier le problème, on suppose qu'un seul type de ressource est nécessaire, mais en quantité différente pour chaque produit.

Les données connues sont :

- n : le nombre de produits différents, $n > 0$
- P_i : les produits fabriqués, P_1, P_2, P_3, … P_n
- Q_i : la quantité de ressource unitaire requise par produit, Q_1, Q_2, Q_3, … Q_n
- Q_t : la quantité de ressource totale disponible
- R_i : les profits (revenus) réalisés par la vente des produits associés, R_1, R_2, R_3, … R_n

Les données recherchées sont les quantités de produits P_i à produire :

- X_i : les produits fabriqués, X_1, X_2, X_3, … X_n

Pour ce problème, c'est à vous de formuler la réponse au problème. Vous devez tenir compte à la fois des profits possibles et des contraintes imposées. De plus, vous devez trouver la réponse à 3 scénarios différents.

Scénario 1 – Boulangerie artisanale.

Quelques informations :

- n = 1
- le temps de cuisson dans le four est la ressource utilisée (en minutes)
- Q_t = 10 heures

i	P_i	Q_i	R_i
1	Pain délicieux	30 minutes	2

Scénario 2 – Entreprise de développement logiciel.

Quelques informations :

- n = 2
- le temps lié au développement est la ressource utilisée (en heures)
- Q_t = 5000 heures

i	P_i	Q_i	R_i
1	Application mobile	200	950
2	Logiciel de bureau	400	2000

Scénario 3 – Entreprise de fabrication de meuble.

Quelques informations :

- $n = 4$
- le bois est la ressource utilisée
- Q_t = 200 unités de bois

i	P_i	Q_i	R_i
1	Table	4 unités de bois	100
2	Chaise	2 unités de bois	60
3	Commode	6 unités de bois	160
4	Étagère	5 unités de bois	120

Pour la résolution de ce problème, le programme doit poser les questions suivantes :

- Quel est le scénario à résoudre : 1, 2 ou 3 (défaut = 3)?

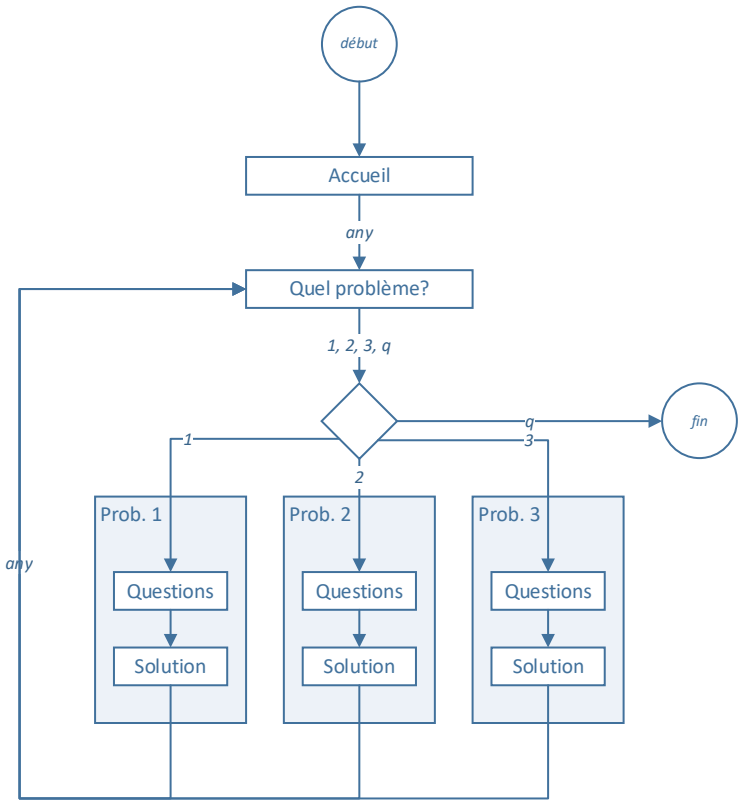
Déroulement du logiciel

Vous devez réaliser un logiciel en mode console présentant le déroulement suivant :

1. Au départ, un message d'accueil du logiciel est affiché. Ce message doit inclure :
   a. Le contexte de réalisation.
   b. Une présentation du projet (en 3 courtes phrases maximum).
   c. Le nom des auteurs.

   On attend l'appui sur une touche quelconque pour passer à l'étape suivante.
2. On affiche un court texte présentant chacun des trois problèmes et on pose la question permettant de savoir lequel des trois problèmes est à résoudre. On offre aussi un quatrième choix permettant de quitter l'application. On attend que l'utilisateur ait fait la saisie.
3. On présente le problème choisi et on pose les questions requises pour le problème. Lorsque toutes les réponses ont été répondues, on passe à l'étape suivante.
4. La simulation correspondante est lancée. Lorsque le résultat est obtenu, les résultats sont affichés. C'est à vous de trouver la meilleure façon de représenter les résultats obtenus. On affiche un texte invitant l'utilisateur à appuyer sur une touche lorsqu'il aura terminé de consulter les résultats pour poursuivre à l'étape suivante.
5. On retourne à l'étape 2.

Le schéma suivant présente le déroulement du programme.



Le mode console permet une interaction simple avec l'usager et il est donc possible de mettre l'emphase sur les parties importantes du projet.

Conception orientée objet

Vous trouverez dans cette section une description semi-détaillée des classes importantes du logiciel. Sachez que toutes les informations ne sont pas présentes et que plusieurs aspects vous reviennent. Néanmoins, **vous devez respecter les structures des éléments présentés** et, lorsque nécessaire, ajouter les parties manquantes que vous jugez nécessaires. Vous devez respecter les noms des classes, des attributs et des méthodes donnés.

Il existe une forte similitude entre la conception orientée objet et la représentation mathématique de l'évolution différentielle faite au début de ce document. Certains choix ont été faits et peuvent être contre-intuitifs à première vue. L'appropriation de ce projet prend un certain temps et l'important est de bien saisir la notion d'abstraction d'un concept destiné à une implémentation modulaire.

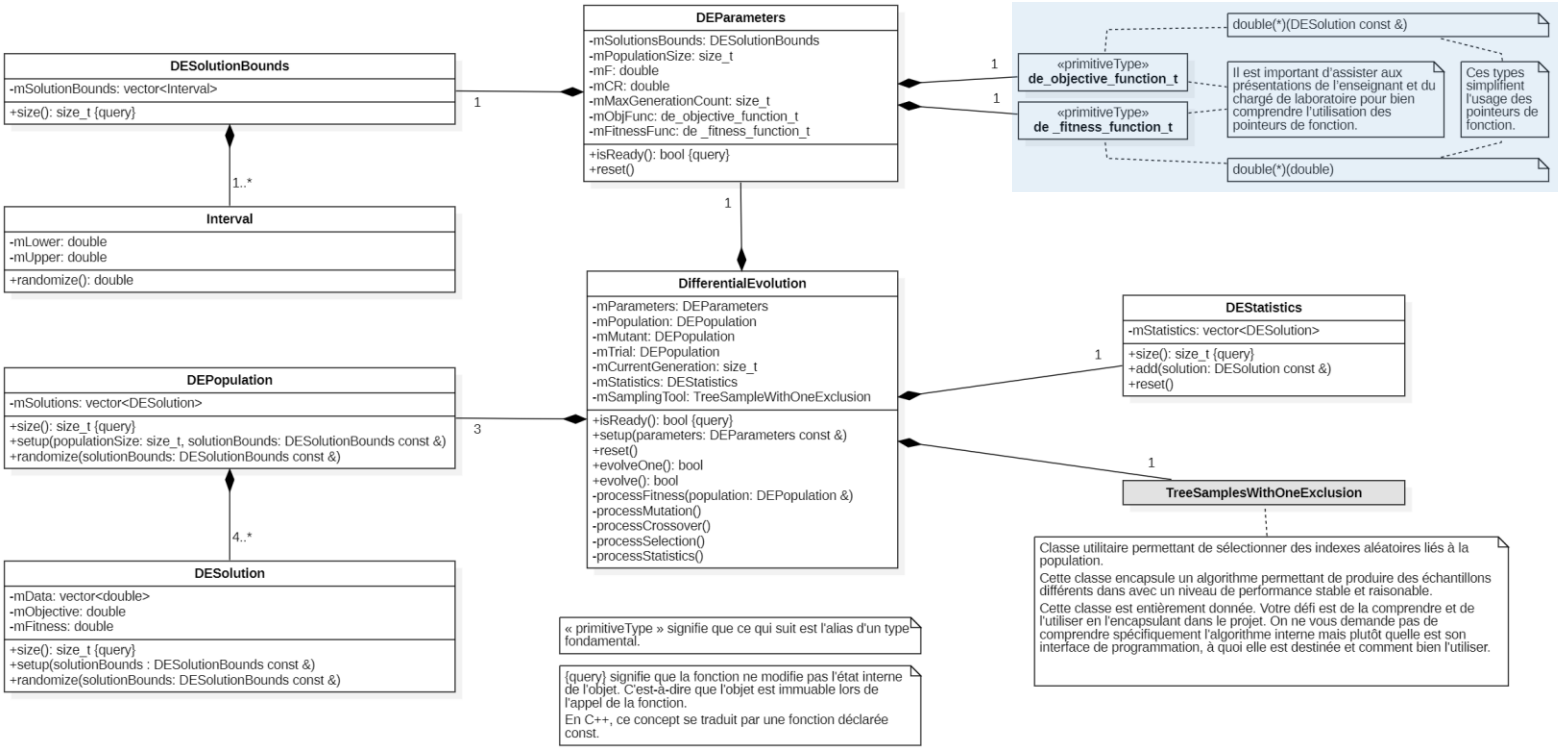
L'approche imposée présente une abstraction intéressante du projet. Trois modules distincts sont présentés :

- L'infrastructure de l'engin d'évolution différentielle;
- Les classes pour ce projet spécifique (interface usager et problématique à résoudre);
- Des outils génériques pouvant être utiles dans n'importe quel projet informatique.

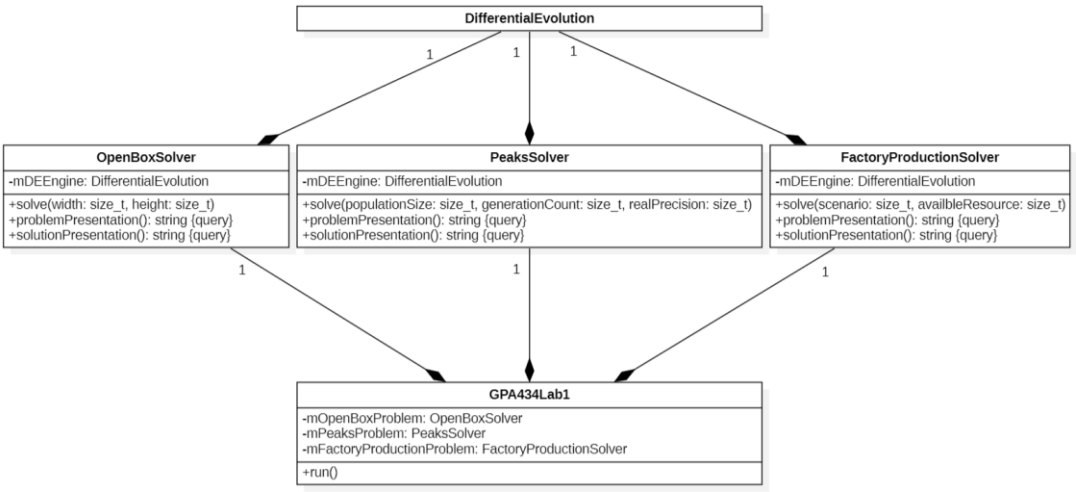
Diagrammes de classes

Un diagramme de classes UML est une représentation visuelle des classes – constituants intra-classe (attributs et opérations) et relations inter-classes (compositions) – utilisées pour modéliser la structure statique d'un logiciel suivant le paradigme de programmation orienté objets.

Attention! Ces diagrammes présentent l'essentiel des infrastructures à mettre en place. Néanmoins, il existe plusieurs autres variables et fonctions non indiquées qui peuvent être essentielles, pertinentes et utiles.



Infrastructure de l'algorithme de l'évolution différentielle



Infrastructure de l'application principale

Détails sur les classes

Lorsque vous étudiez pour la première fois un schéma de conception, il est important de bien comprendre le sens donné à chaque élément. Dans ce cas-ci, il est essentiel de bien saisir le rôle et mandat de chaque classe, mais aussi des constituants internes. Vous ne pouvez pas réaliser ce projet sans comprendre la structure imposée.

Voici les grandes lignes du schéma de conception. Assurez-vous de bien le comprendre. Ça vous aidera à répartir la charge de travail dans l'équipe, à faire des tests bien ciblés, à fusionner votre travail et à le déboguer. N'hésitez surtout pas à poser des questions au chargé de laboratoire.

`Interval`

La classe `Interval` représente le concept mathématique d'un intervalle de nombres réels. On appelle intervalle un ensemble de nombres délimité par deux nombres réels constituant une borne inférieure et une borne supérieure.

- `double mLower` représente la borne inférieure de l'intervalle
- `double mUpper` représente la borne supérieure de l'intervalle
- `double randomize()` retourne un nombre aléatoire inclus dans l'intervalle

Pour ce projet, les intervalles sont utilisés pour représenter le domaine de valeur de chaque dimension du problème à résoudre. Pour simplifier le projet, on considère les bornes systématiquement inclusives. Il est important de s'assurer que les bornes ne soient pas permutées, c'est-à-dire que la borne inférieure soit plus petite ou égale à la borne supérieure.

`DESolutionBounds`

Cette classe représente $\vec{l}$ et $\vec{u}$. Elle offre tous les services nécessaires à la gestion adéquate des informations requises. Ainsi, l'utilisation d'un vecteur d'intervalle permettra à la fois de connaître la dimension du problème ainsi que les bornes supérieures et inférieures de chaque paramètre (de chaque dimension).

- `std::vector<Interval> mSolutionBounds` représente tous les intervalles
- `size_t size() const;` retourne le nombre d'intervalles, soit la taille du problème

Pour ce projet, cette classe représente l'un des éléments fondamentaux du système : l'architecture des solutions. On y retrouve la dimensionnalité du problème et le domaine pour chaque dimension.

`DESolution`

Représente une solution candidate $\vec{s}$. Les candidats sont des vecteurs de nombres réels représentant un point dans l'espace de solution Θ.

- `std::vector<double> mData` représente le vecteur de données
- `double mObjective` le résultat du calcul de la fonction objective
- `double mFitness` le résultat du calcule de « fitness »
- `size_t size() const;` retourne le nombre de paramètres, soit la taille du problème
- `void setup(DESolutionBounds const & solutionBounds)` dimensionne le vecteur de données à la dimensionnalité du problème
- `void randomize(DESolutionBounds const & solutionBounds)` assigne à chacune des données du vecteur une valeur générée aléatoirement par chacune des intervalles de `solutionBounds`

`DEPopulation`

Représente un ensemble de solution. Cette classe est une classe utilitaire permettant de gérer plus facilement toutes les solutions dans un tout cohérent. Cette classe est importante car trois instances seront nécessaires : $\vec{x}$, $\vec{m}$ et $\vec{t}$.

- `std::vector<DESolution> mSolutions` le vecteur de solutions
- `size_t size() const;` retourne le nombre de solutions (la taille de la population)
- `void setup(size_t populationSize, DESolutionBounds const & solutionBounds)` la fonction faisant l'initialisation de la population. D'abord le vecteur est redimensionné à `populationSize`. Ensuite chaque solution est initialisée par l'appel de `setup`.
- `void randomize(DESolutionBounds const & solutionBounds)` la fonction générant des valeurs aléatoires pour toute la population.

`DEParameters`

Ceci est un conteneur d'informations. Cette classe contient tous les paramètres nécessaires au déroulement de l'algorithme. C'est cette classe qui fait le pont entre la librairie générique `DE` et le logiciel spécifique que vous avez à faire.

Pour solutionner les trois problématiques données, c'est ici que vous préciserez chacune des spécificités nécessaires à la résolution de la problématique.

- `DESolutionBounds mSolutionBounds` contient la structure d'information définissant le domaine de chaque paramètre du problème.
- `size_t mPopulationSize` le taille de la population *P* (la taille des vecteurs $\vec{x}$, $\vec{m}$ et $\vec{t}$).
- `double mF` la valeur de la constante *F*.
- `double mCR` la valeur de la constante *CR*.
- `double mMaxGenerationCount` le nombre de générations avant d'arrêter.
- `de_objective_function_t mObjFunc` la fonction objective.
- `de_fitness_function_t mFitnessFunc` la fonction « *fitness* ».

Il est important d'assister aux présentations de l'enseignant et du chargé de laboratoire pour bien comprendre l'utilisation des pointeurs de fonction (`de_objective_function_t` et `de_fitness_function_t`).

`DifferentialEvolution`

`DifferentialEvolution` est la classe fondamentale de l'algorithme à implémenter. C'est elle qui implémente les différents algorithmes évolutifs et concentre toutes les classes utilitaires développées jusqu'ici.

Son rôle est de contenir les trois populations (la population courante $\vec{x}$, les mutants $\vec{m}$ et les tests $\vec{t}$) et de structurer la démarche algorithmique. La fonction `evolveOne` réalise un pas d'évolution et réalise toutes les opérations pour passer à la génération suivante.

Plusieurs fonctions sont privées et sont des outils internes pour fragmenter et simplifier le code relié à `evolveOne`.

- `bool isReady() const` fonction validant que tous les paramètres sont viables pour faire l'évolution.
- `void setup(DEParameters const & parameters)` fonction réalisant l'initialisation complète de l'algorithme évolutif.
- `void reset()` réinitialise l'algorithme évolutif à son état initial et le rend apte à reprendre une autre simulation.
- `bool evolveOne()` réalise un pas de simulation et passe à la génération suivante. Cette fonction appelle les fonctions privées permettant de réaliser l'évolution différentielle.
- `bool evolve()` réalise tous les pas de simulation. Cette fonction appelle en boucle `evolveOne` afin d'atteindre G_{max}.
- `void processFitness(DEPopulation & population)` calcule la « fitness » sur toutes les solutions de la population donnée.
- `void processMutation()` calcule le vecteur des mutants.
- `void processCrossover()` calcule le vecteur des tests.
- `void processSelection()` calcule le vecteur courant et détermine la nouvelle génération.
- `void processStatistics()` calcule les statistiques de la génération courante.

`DEStatistics`

Cette classe est un conteneur d'informations. Elle contient un vecteur de `DESolution` de la taille du nombre de générations en cours où est stockée la meilleure solution de chaque génération. C'est l'historique d'évolution.

- `std::vector<DESolution> mStatistics` la meilleure solution pour chaque génération (chaque évolution).
- `void add(DESolution & const solution);` ajoute une solution à la fin
- `void reset()` réinitialise à vide le vecteur d'information.

`OpenBoxSolver`

Classe implémentant la résolution du problème de la boîte ouverte.

`PeaksSolver`

Classe implémentant la résolution du problème de la fonction « *peaks* » de Matlab.

`FactoryProductionSolver`

Classe implémentant la résolution du problème de production manufacturière.

`ThreeSamplesWithOneExclusion`
Classe générant un échantillonnage à trois valeurs exclusives. Cette classe vous est donnée. Lire la documentation à même le code source pour les détails et savoir comment l'utiliser.

`GPA434Lab1`
Classe représentant le cœur du programme. On y retrouve la boucle principale de l'application, la gestion de l'interaction avec l'utilisateur et des appels liés à la résolution de problème ainsi que de l'affichage des résultats. Évidemment, cette classe peut être subdivisée en plusieurs classes utilitaires. C'est à vous d'en faire un découpage adéquat.

Contraintes

Voici les contraintes du projet :

- Contraintes techniques :
  - Vous devez respecter les noms des types, classes, attributs et méthodes données.
  - Vous devez respecter la norme de codage imposée.
  - Il est attendu que votre code soit documenté adéquatement.
  - Votre code ne doit pas utiliser :
    - l'instruction `goto`;
    - des variables globales;
    - des littéraux sans variables symboliques (sauf lorsque trivial).
  - Il est obligatoire de réaliser le projet avec le langage de programmation C++ et l'environnement Visual Studio. Dans les deux cas, les versions les plus récentes sont encouragées.
- Contraintes liées au travail et à la remise :
  - Vous avez quatre périodes pour réaliser ce laboratoire.
  - Vous devez obligatoirement être en équipe de deux ou trois étudiants (on encourage des équipes de 2 étudiants). L'objectif est d'échanger autant que possible sur les notions de l'orienté objet et de mettre en pratique le travail collaboratif cher en ingénierie. **Aucun travail réalisé seul ne sera accepté.**
  - Sans faire de plagiat, l'entre aide entre les équipes est encouragée.
  - Assurez-vous de bien comprendre les éléments que vous remettez. Si le chargé de laboratoire vous pose une question sur votre travail et que vous n'êtes pas capable de répondre ou d'expliquer ce dernier, vous aurez systématiquement une note de zéro pour plagiat. Une telle situation suivra le protocole de discipline de l'ÉTS prévu à cet effet.
  - Il est attendu que la qualité générale du rapport final soit du niveau d'un étudiant en génie. La qualité du français est évaluée et vous pouvez perdre jusqu'à 10 %.

Fichiers mis à votre disposition

Les fichiers suivants sont mis à votre disposition :

- `ThreeSamplesWithOneExclusion.[h/cpp]`
  La mutation réalisée par l'évolution différentielle requiert un algorithme permettant de

faire la génération de valeurs aléatoires distinctes. Ces deux fichiers vous donnent une classe réalisant cette tâche. Vous trouverez la documentation nécessaire à l'utilisation de la classe à même le fichier d'en-tête.

Démarche suggérée

La démarche suivante vous aidera à mieux cibler l'ordre des tâches à faire pour réaliser un tel projet.

1. Avant de commencer :
    a. Formez les équipes
    b. Assurez-vous d'avoir fait une lecture de l'énoncé.
    c. Assurez-vous de bien comprendre les éléments techniques présentés.
2. Démarrage du projet :
    a. Créez un projet dans Visual Studio.
    b. Ajoutez au projet les fichiers fournis :
        i. `ThreeSamplesWithOneExclusion.h/cpp`
    c. Ajoutez au projet les nouveaux fichiers suivants :
        i. `main.cpp`
        ii. `rapport.txt`
3. Développement des classes :
    a. Assurez-vous de bien comprendre le rôle de chaque classe à développer afin de cerner les tâches nécessaires à réaliser.
    b. Développez de petites fonctions modulaires (DRY) et faites rapidement de petits tests validant votre travail au fur et à mesure (CALTAL).
    c. Voici une suggestion concernant l'ordre du développement à réaliser. Même s'il existe des dépendances fortes, il existe trois tâches principales à faire dont une grande partie peut être réalisée en parallèle :
        i. Développement 1 – Librairie DE :
            1. `Interval`
            2. `DESolutionBounds`
            3. `DESolution`
            4. `DEPopulation`
            5. `DEParameters`
            6. `DifferentialEvolution`
            7. `DEStatistics`
        ii. Développement 2 – Résolution de problèmes :
            1. `OpenBoxSolver`
            2. `PeaksSolver`
            3. `FactoryProductionSolver`
        iii. Développement 3 – Application principale et interface usager :
            1. `GPA434Lab1`
4. Faires les tests globaux.
5. Rédaction du rapport.

Rapport

On vous demande d'ajouter un fichier texte à même la solution de Visual Studio dans lequel vous répondrez à ces questions. :

1. Donnez les noms de tous les étudiants impliqués dans le projet.
2. Fonctionnalité :
    a. Avez-vous réussi à produire adéquatement l'algorithme de l'évolution différentielle? S'il y a des parties incomplètes ou non fonctionnelles, quelles sont-elles.
    b. Pour chacun des trois problèmes donnés, indiquez :
        i. Modélisation du problème :
            1. Quel est le nombre de dimensions?
            2. Pour chaque dimension, expliquer ce qu'elle représente, son domaine et son unité (s'il existe).
            3. Y a-t-il des contraintes à même la formulation du problème?
            4. Quelle est la fonction objective?
            5. Quelle est la fonction de « *fitness* » ?
        ii. Implémentation de la solution :
            1. Avez-vous terminé son implémentation?;
            2. Convergent-elle vers une solution viable?;
            3. Donnez le résultat obtenu par une simulation qui utilise les valeurs par défaut.
3. Discussions :
    a. Quel est l'impact de varier les paramètre *F* et *CR* dans le premier problème?
    b. Quel est l'impact de varier les paramètre *P* et G_{max} dans le deuxième problème?
    c. Expliquer quel est l'impact du paradigme l'orienté objet sur le développement de ce projet. Faites une analyse comparative entre les paradigmes de programmation procédurale (que vous avez utilisé avec la programmation en langage C) et orienté objets.
    d. Serait-il possible d'améliorer la modularité de ce projet? Comment?
4. Avez-vous des commentaires constructifs à faire sur ce projet (coquilles et suggestions).

Ce fichier doit être nommé : `rapport.txt`.

Ce document est **obligatoire** pour que votre travail soit corrigé.

Remise

La remise se fait sur Moodle :

- avant la cinquième période de laboratoire (au début de la cinquième période, vous aurez une pénalité équivalente à un jour de retard si votre remise n'est pas faite);
- une remise par équipe;
- Le document ZIP contenant votre travail :
    - nom du document ZIP : `GPA434_Lab1_NomEtudiant1_NomEtudiant2[_NomEtudiant3].zip`
    - le document ZIP doit contenir votre solution tout **en vous assurant de faire le nettoyage** de votre solution dans Visual Studio.

Les instructions de la prochaine section indiquent comment le nettoyage de votre solution. Il est très important de faire ce nettoyage avant la remise.