ENACOM

DA CIÊNCIA AO PRODUTO

Sexta of Python

Framework de otimização - 14/05/2021

Agenda

- Visão geral
- Arquitetura do framework
- Exemplos

Visão geral

Visão geral

- Algoritmos determinísticas para otimização:
  - Linear;
  - Não linear;
  - Multiobjetivo.

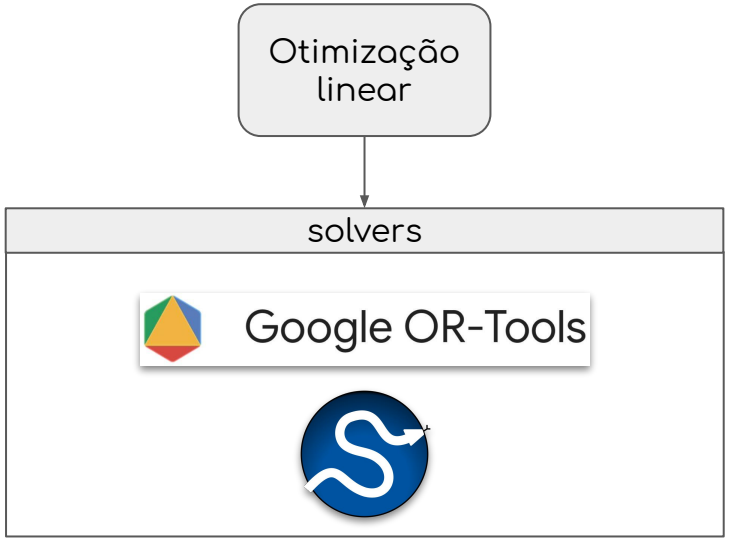
Visão geral

- Algoritmos determinísticas para otimização:
  - Linear;
  - Não linear;
  - Multiobjetivo.
- Algoritmos próprios (contribuição científica):
  - Elipsoidal multiobjetivo;
  - Nondominated sampler;
  - Seção áurea multimodal.

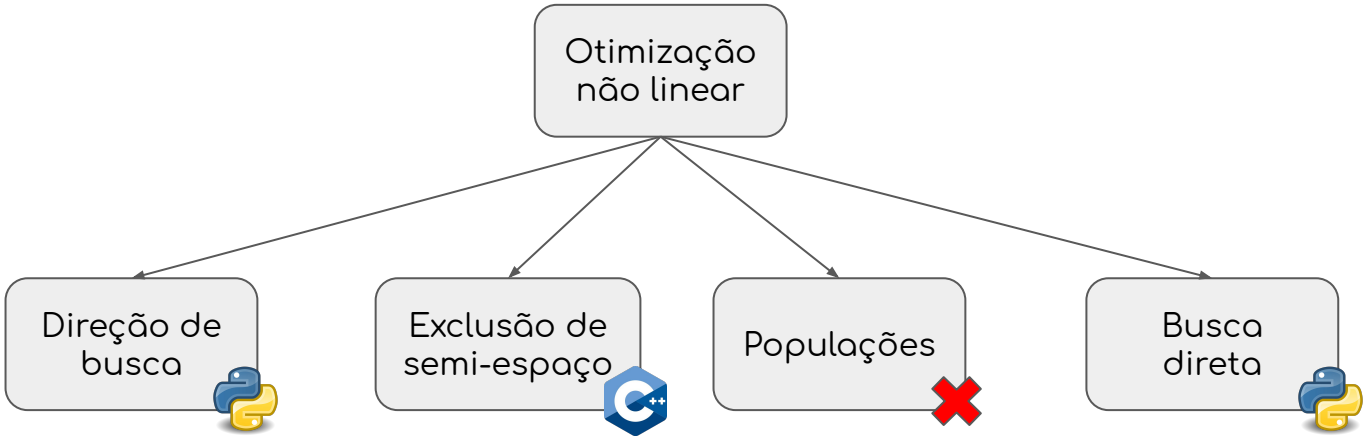
Visão geral

- Algoritmos determinísticas para otimização:
  - Linear;
  - Não linear;
  - Multiobjetivo.
- Algoritmos próprios (contribuição científica):
  - Elipsoidal multiobjetivo;
  - Nondominated sampler;
  - Seção áurea multimodal.
- Repositório: **science-optimization**

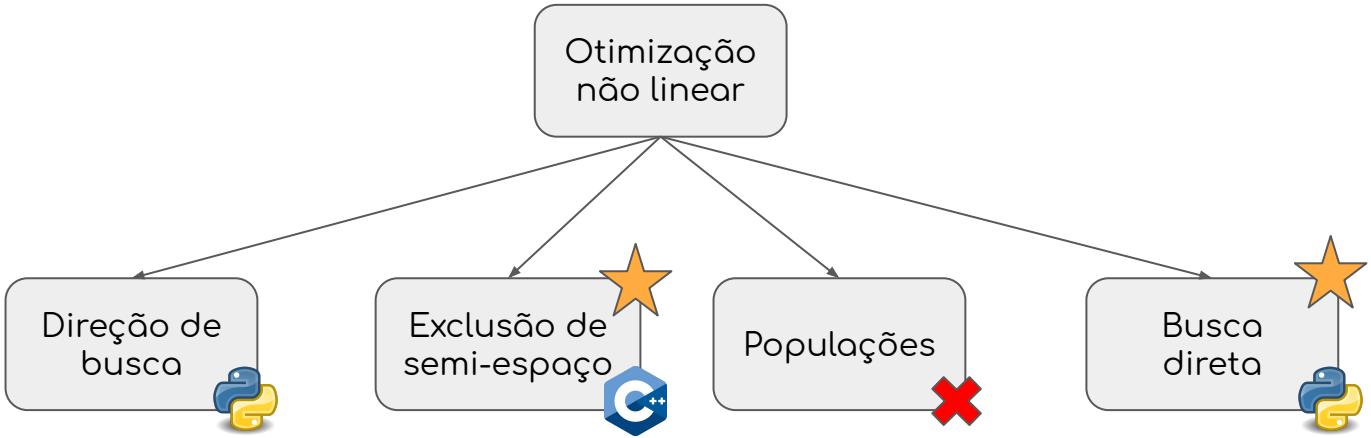
Solvers lineares



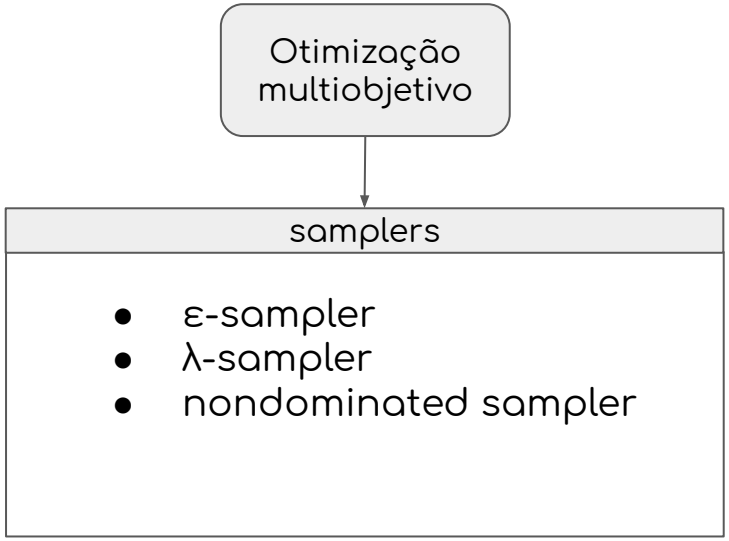
Otimização não linear



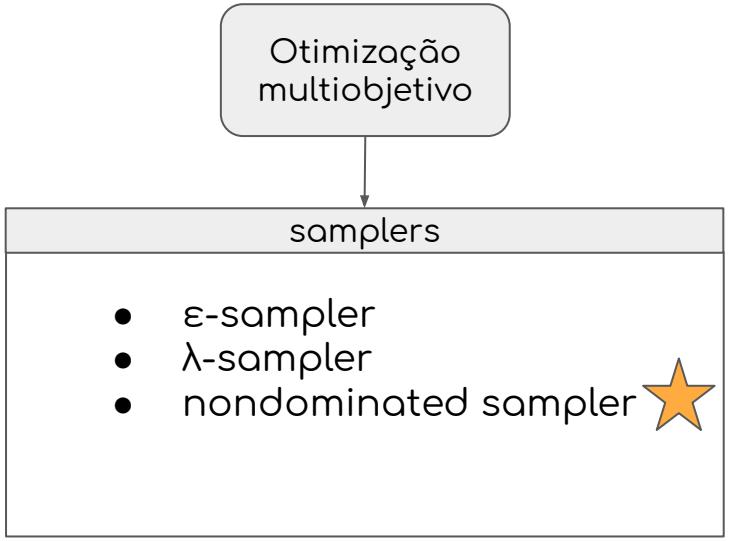
Otimização não linear



Otimização multiobjetivo



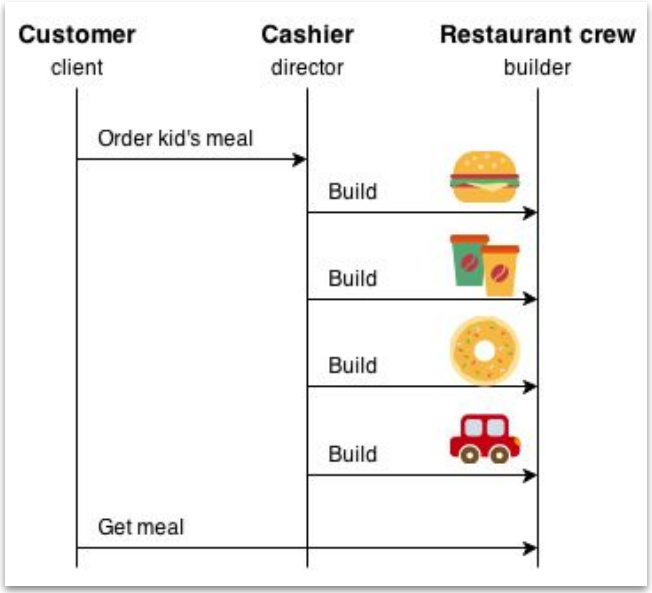
Otimização multiobjetivo



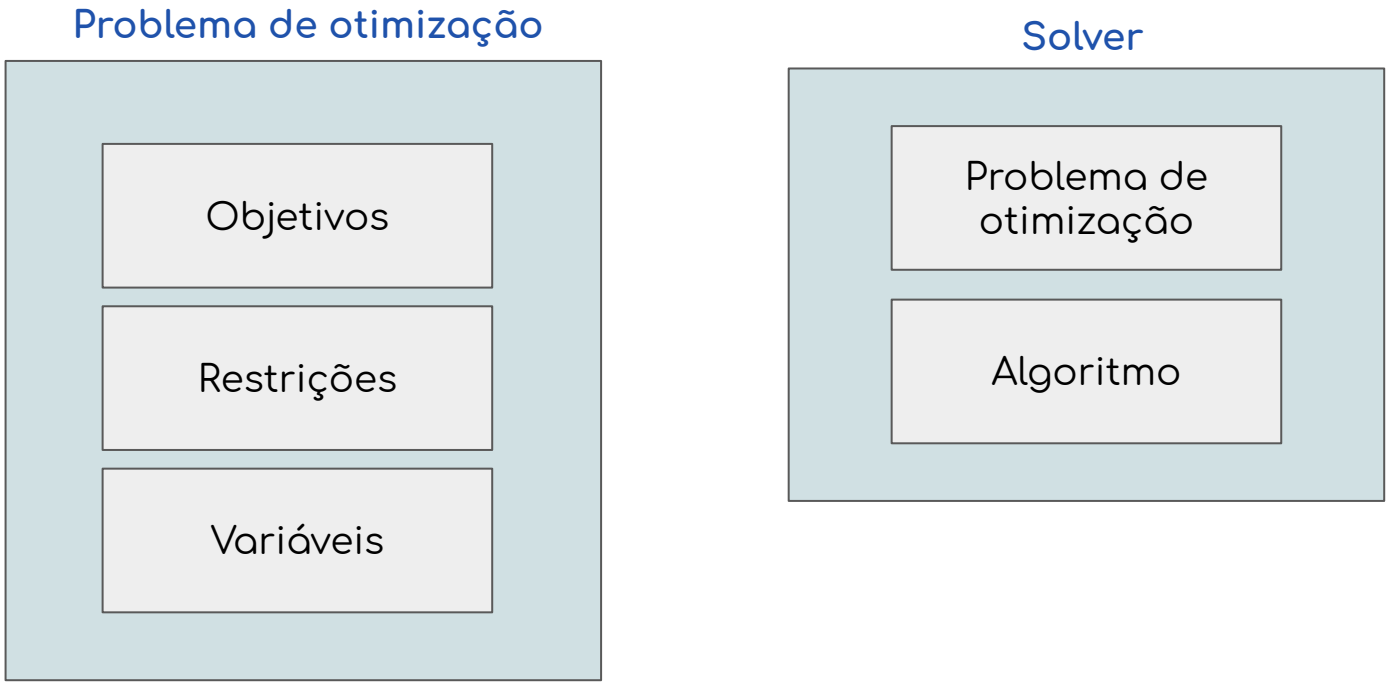
Arquitetura

Arquitetura do framework

Builder



Arquitetura do framework



Arquitetura do framework

```python
class BuilderOptimizationProblem(metaclass=abc.ABCMeta):
    """
    This class is responsible for constructing all the parts of a problem.
    """

    @abc.abstractmethod
    def build_objectives(self):
        pass

    @abc.abstractmethod
    def build_constraints(self):
        pass

    @abc.abstractmethod
    def build_variables(self):
        pass
```

Arquitetura do framework

```python
class Diet(BuilderOptimizationProblem):
    """Concrete builder implementation.

    This class builds a diet problem.

    """

    def __init__(self, food, food_limits, cost, nutrient, demand):
        """ Constructor of Diet optimization problem.

        Args:
            food: food names
            food_limits: servings limit
            cost: cost per serving
            nutrient: nutrients on each food
            demand: nutrients demand
        """
        self.food = food
        self.food_limits = food_limits
        self.cost = cost
        self.nutrients = nutrient
        self.demand = demand
```

Arquitetura do framework

```python
def build_variables(self):

        # add variables
        variables = Variable(x_min=self.food_limits[:, 0].reshape(-1, 1),
                             x_max=self.food_limits[:, 1].reshape(-1, 1))

        return variables
```

```python
    def build_objectives(self):

        # add objective
        obj_fun = FunctionsComposite()
        obj_fun.add(f=LinearFunction(c=1.*np.array(self.cost).reshape(-1, 1)))
        objective = Objective(objective=obj_fun)

        return objective
```

```python
    def build_constraints(self):

        # parameters
        c = np.vstack((np.array(self.nutrients), -1.0 * np.array(self.nutrients)))
        d = np.vstack((-1.0 * np.array(self.demand)[:, 1].reshape((-1, 1)),
                       np.array(self.demand)[:, 0].reshape((-1, 1))))

        # add constraint
        ineq_cons = FunctionsComposite()
        for i in range(c.shape[0]):
            ineq_cons.add(f=LinearFunction(c=c[i, :].reshape((-1, 1)), d=d[i]))

        constraints = Constraint(ineq_cons=ineq_cons)

        return constraints
```

Arquitetura do framework

```python
class OptimizationProblem:
    """Optimization problem class.

    Canonical form:
        minimize f(x)
        s.t.     g(x) <= 0
                 h(x) = 0

    An optimization problem is assembled by the Model class from parts
    made by Builder. Both these classes have influence on the resulting
    object.

    """

    def __init__(self, builder: BuilderOptimizationProblem):
        """ Constructor of an optimization problem.

        Args:
             builder: (BuilderOptimizationProblem) class Builder instance
        """

        # builder problem
        self._build_problem(builder)
```

```python
def _build_problem(self, builder):
    """Build an optimization instance

    Args:
        builder: class Builder instance

    """

    # problem attributes
    variables = builder.build_variables()
    objective = builder.build_objectives()
    constraints = builder.build_constraints()

    # consistency check
    self._check_consistency(variables=variables,
                            objectives=objective,
                            constraints=constraints)

    # builder problem
    self.variables = variables
    self.objective = objective
    self.constraints = constraints
```

Arquitetura do framework

```python
def diet_example():

    # options
    food = ['corn',
            'milk',
            'white-bread']

    # minimum and maximum servings of each food
    food_limits = np.array([[0, 10],
                            [0, 10],
                            [0, 10]])

    # cost of each food
    cost = np.array([.18, .23, .05])

    # minimum and maximum demand of each nutrient
    demand = np.array([[2000, 2250],
                       [5000, 50000]])

    # nutrients on each food
    nutrient = np.array([[72, 121, 65],
                         [107, 500, 0]])

    # builder diet instance
    diet_problem = OptimizationProblem(builder=Diet(food=food,
                                                    food_limits=food_limits,
                                                    cost=cost,
                                                    nutrient=nutrient,
                                                    demand=demand))

    # builder optimization
    optimizer = Optimizer(opt_problem=diet_problem, algorithm=ScipySimplexMethod())
    results = optimizer.optimize()
```

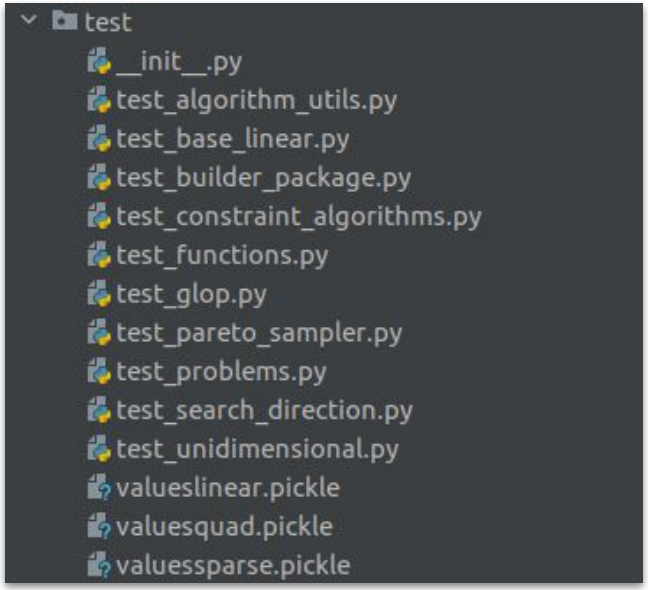
Arquitetura do framework

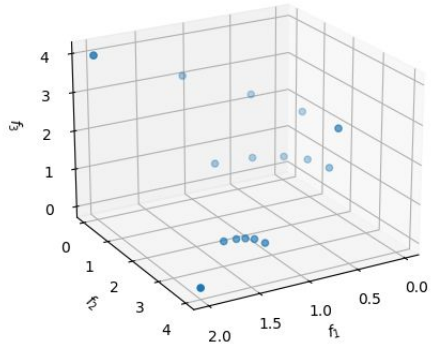
```python
class BaseFunction(metaclass=abc.ABCMeta):
    """
    Base Function.
    """


    @abc.abstractmethod
    def eval(self, *args):
        pass

    def gradient(self, x):
      # code here

    def hessian(self, x):
      # code here
```

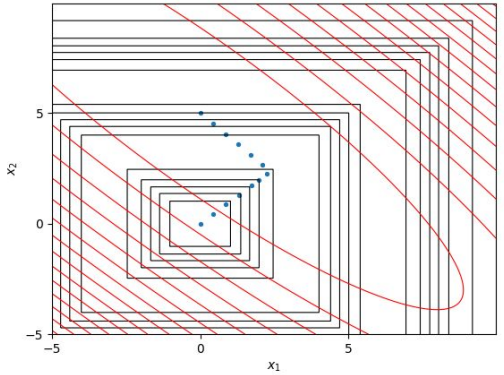
```python
# overloading add operator
def __add__(self, other):
    # code here

# overloading subtraction operator
def __sub__(self, other):
    # code here

# overloading multiplication operator
def __mul__(self, other):
  # code here

# overloading divide operator
def __truediv__(self, other):
  # code here
```

Arquitetura do framework

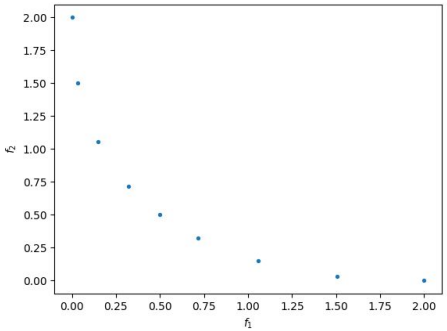
>90% de cobertura de código!

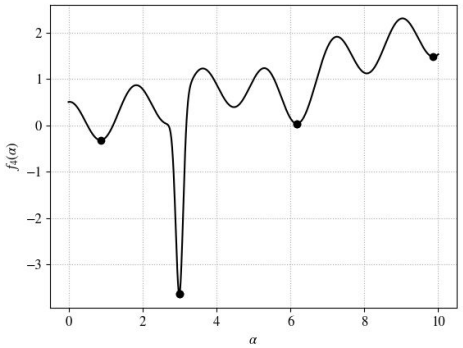



4
3
2
1
0
f3
0
1
2
3
4
f2
2.0
1.5
1.0
0.5
0.0
f1


5
0
−5
x2
−5
0
5
x1

Exemplos


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
f2
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
f1


2
1
0
−1
−2
−3
f1(α)
0
2
4
6
8
10
α

ENACOM

DA CIÊNCIA AO PRODUTO