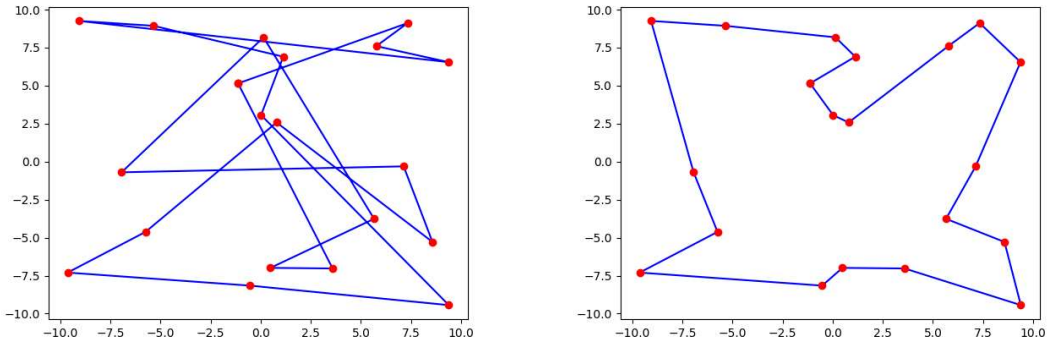
Problemas de Otimização

Problema 1: Problema do caixeiro viajante (*travelling salesman problem* - TSP).

A entrada do TPS é um conjunto de pontos representando posições de cidades. A solução ótima é o trajeto mais curto para, partindo de uma cidade inicial, visitar todas as cidades uma vez e depois retornar à cidade inicial. A figura 1 exemplifica duas soluções para o TSP para um conjunto de pontos dado, sendo que a figura 1 (a) é uma solução com maior custo que aquela apresentada na figura 1 (b). Para construção do relatório deve ser usado o arquivo com 30 pontos. Os arquivos contêm as coordenadas (x, y) de um ponto por linha. Assuma que a cidade inicial do trajeto é dada pelo primeiro ponto do arquivo.

- **Indivíduo**: Uma solução para o problema será dada por uma sequência de números inteiros representando a ordem em que as cidades serão visitadas. Como a primeira cidade da lista é sempre o início e fim do trajeto, não é necessário adicionar ela ao estado (mas ela deve ser considerada no cálculo da função objetivo). Como cada cidade deve ser visitada apenas uma vez, não devem existir repetições de cidades no estado.
- **Função Objetivo**: A função objetivo a ser minimizada é o custo total do trajeto.
- **Mutação**: Troca da posição de duas cidades escolhidas aleatoriamente no trajeto. Por exemplo, o estado [2, 4, 5, 6, 3] poderia ser transformado em [2, 6, 5, 4, 3] trocando as posições das cidades 4 e 6.
- **Crossover**: Order 1 Crossover.
- **Parâmetros sugeridos**: População de 200 indivíduos, 1000 gerações, probabilidade de mutação entre 10% e 20%.



(a) (b)

Figura 1: As figuras apresentam soluções para o problema do caixeiro viajante (*travelling salesman problem* – TSP). A solução (a) tem custo de caminho maior que a solução (b). Observe que o número de cruzamentos de trechos em (b) é menor que em (a).

Problema 2: Regressão.

Imagine que você trabalha no departamento de data science de uma empresa que deseja encontrar uma função que descreve como um determinado fenômeno acontece. Para isto, foram coletados dados sobre este fenômeno por algum tempo. Infelizmente, devido à imperfeição dos sensores, os dados foram medidos com um pouco de ruído.

Você foi escolhido para encontrar a função que descreva a evolução dos valores. A figura 2 (a) mostra as medições que você recebeu e a figura 2 (b) traz em vermelho a função verdadeira (e desconhecida) que gerou os pontos. A sua tarefa é encontrar uma função que aproxime a função da figura 2 (b). Para isto, utilizaremos regressão de funções periódicas. Neste tipo de regressão, tentamos fazer a aproximação usando funções com o seguinte formato:

$$f(x) = \boldsymbol{a_0} + \sum_{i=1}^{n} \boldsymbol{a_i} \sin(i * x) + \boldsymbol{b_i} * \cos(i * x) \qquad (1)$$

onde n define o número de componentes, e $\boldsymbol{a_i}$ e $\boldsymbol{b_i}$ são coeficientes a serem calculados usando o algoritmo de otimização. Assumiremos que o número de componentes n = 4. A figura 3 ilustra o formato das funções para diferentes valores dos coeficientes. A figura no canto inferior direito é considerada uma boa aproximação da função verdadeira.

- **Indivíduo**: 9 valores reais representando os coeficientes $\boldsymbol{a_0}$, e $\boldsymbol{a_i}$ e $\boldsymbol{b_i}$ para os 4 componentes.
- **Função Objetivo**: A função objetivo será dada pelo erro médio quadrático. Seja P = [(x1, y1), ..., (xn, yn)] o conjunto de pontos dados como entrada. Então, a função objetivo será:

$$E = \frac{1}{N} \sum_{(xi,yi) \in P} \left(yi - f(xi)\right)^2 \qquad (2)$$

- **Mutação**: Em 50% dos casos, atualize o valor de um coeficiente para um valor aleatório no conjunto de valores possíveis. Nos outros casos, some ao valor de um coeficiente um valor aleatório amostrado de uma distribuição normal padrão (média 0 e desvio padrão 1).
- **Crossover**: Sejam p1 e p2 dois pontos escolhidos para produzirem filhos. A operação de crossover será dada pela média ponderada c1 = p1 α + p2 (1 - α) e c2 = p2 α + p1 (1 - α), onde α é um número aleatório amostrado de uma distribuição uniforme entre 0 e 1.
- **Parâmetros:** Taxa de mutação entre 10% e 20%. Intervalo de valores dos coeficientes: [-100, 100]. Tamanho da população: 200. Número de gerações: 1000.

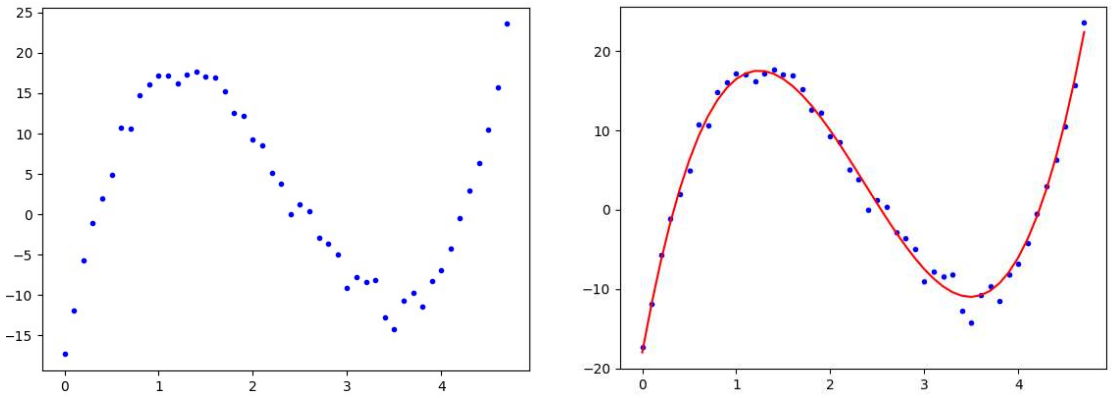


Figura 2: (a) Pontos medidos usando um sensor. (b) sobreposição dos pontos medidos sobre a função verdadeira (e desconhecida) que descreve os dados.

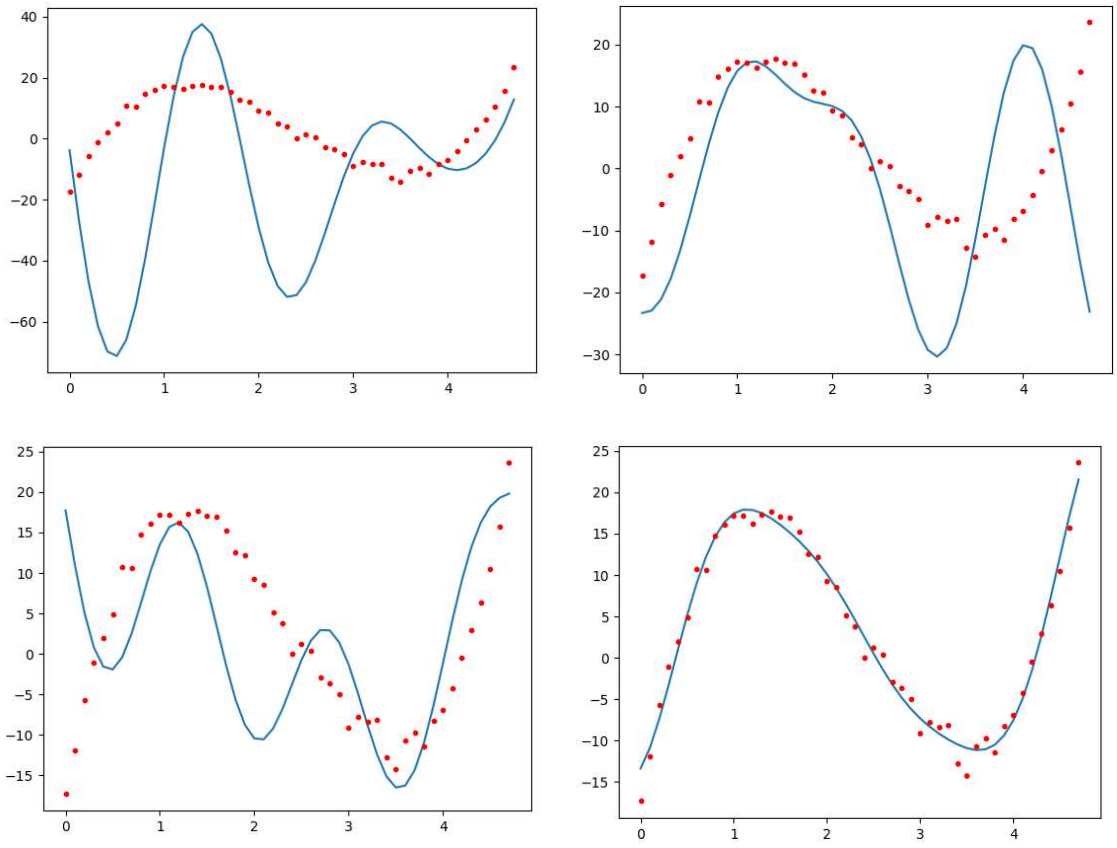


Figure 3: Exemplos de funções que podem ser obtidas variando os valores dos coeficientes na função apresentada na equação 1 considerando n = 4.

Problema 3: Busca de hiperparâmetros e feature selection.

Suponha que você trabalha para um banco que deseja construir um modelo capaz de predizer se uma pessoa irá pagar um empréstimo ou não de forma que o banco para decidir *a priori* aprovar ou negar empréstimos analisando o perfil dos clientes. Para isto, foi construída uma base de dados e a sua tarefa é identificar o subconjunto de características e os hiperparâmetros dos modelos que levam ao modelo mais preciso. A base de dados que será utilizada é a German Credit Data Statlog Numeric (link abaixo). O algoritmo de classificação será o *k-nearest neighbors* (K-NN). Os hiperparâmetros a serem buscados são os valores de K e a métrica de distância a ser utilizada.

Base de Dados: https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

- **Indivíduo**: O indivíduo será híbrido e será composto por padrão binário representando quais features serão consideradas e dois valores categóricos representando o número de vizinhos K e a métrica usada para comparar indivíduos (ver parâmetros).
- **Função Objetivo**: Treinar o classificador com os valores do indivíduo e medir a acurácia (número de acertos sobre total de amostras) no conjunto de validação. Soluções para as quais o padrão de bits é completamente 0 (todas as features excluídas) são consideradas inváidas. Neste caso, retorne um valor muito alto para a função objetivo.
- **Mutação**: Inverter um bit aleatório ou escolher um valor aleatório para K dentre os valores possíveis ou escolher aleatoriamente uma métrica dentre os valores possíveis.

- **Crossover**: No padrão binário, usar crossover de 1 ponto (quem quiser pode usar o crossover de 2 pontos). Nos valores categóricos, faremos um crossover simples. No primeiro filho, o valor de K virá do primeiro pai e o valor da métrica do segundo pai. No segundo filho, faremos o contrário. O valor de K virá do segundo pai e o valor da métrica do primeiro pai.
- **Parâmetros:** Valores de K possíveis: {1, 3, 5, 7, 9, 11, 13, 15}. Métricas possíveis: {"euclidean", "hamming", "canberra", "braycurtis"}. Tamanho da população: 200. Número de gerações: 500.
- **OBS**: Esse problema vai demorar um pouco para rodar. Se estiver inviável, reduza o tamanho da população para 100 e o número de gerações para 250.

Informações Úteis

- Comando para instalar as bibliotecas numpy (computação numérica), matplotlib (gráficos) e seaborn (análise estatística e gráficos mais bonitos que a matplotlib) e scikit-learn (machine learning):

```
pip install numpy matplotlib seaborn sklearn
```

- numpy.random.uniform: gera números aleatórios em um intervalo usando uma distribuição uniforme.
- numpy.random.uniform: gera números aleatórios usando uma distribuição normal.
- numpy.random.randint: função para produzir um ou mais números inteiros aleatórios.
- numpy.random.choice: escolhe um valor aleatório em uma coleção.
- numpy.concatenate: cria um array que é dado pela concatenação de outros.
- matplotlib.pyplot.plot(x, y, "-b"): cria um gráfico com pontos cujas coordenadas são dados pelos arrays x e y conectando os pontos usando uma reta azul ("-" indica que os pontos devem ser conectados e "b" representa "blue").
- matplotlib.pyplot.plot(x, y, ".r"): cria um gráfico com pontos cujas coordenadas são dados pelos arrays x e y exibindo apenas os pontos em vermelho (".” indica os pontos não devem ser conectados e "r" representa "red").
- Galerias de gráficos com código mostrando como produzi-los:
  Matplotlib: https://matplotlib.org/stable/gallery/index.html
  Seaborn: https://seaborn.pydata.org/examples/index.html
- matplotlib.pyplot.show: mostra o gráfico na tela e o programa pausa até que a janela seja fechada. Para saber como exibir o gráfico sem bloquear o programa veja o link abaixo.
  Link:https://stackoverflow.com/questions/28269157/plotting-in-a-non-blocking-way-with-matplotlib#:~:text=import%20numpy%20as,__name__%20%3D%3D%20%27__main__%27%3A%0A%20%20%20%20main() .
- copy.deepcopy: cria uma cópia completa de um objeto (a função é chamada recursivamente para atributos que também sejam objetos).