SMAP para Bacia de Camargos - Calibração e Validação

Carregar os dados

In [3]:
```
import pandas as pd
import warnings; warnings.filterwarnings('ignore')

# df = pd.read_csv('data/smap_input.csv')
df = pd.read_csv('data/bacia-camargos.csv')

# Data cleaning
df['Ep'] = df['Ep'].str.replace(',', '.').astype('float')
df['Pr'] = df['Pr'].str.replace(',', '.').astype('float')

df.set_index(pd.to_datetime(df['data']), inplace=True)
df.drop('data', axis=1, inplace=True)

df.head()
```

Out[3]:

	Qobs	Ep	Pr
data			
1995-01-01	204	4.94	4.3
1995-01-02	181	4.94	9.1
1995-01-03	176	4.94	22.8
1995-01-04	194	4.94	9.2
1995-01-05	198	4.94	1.7

Modelo Base

Parâmetros usando o meio dos intervalos

In [4]:
```
params_middle = dict(
    Str = 1050,
    Crec = 50,
    Capc = 40.0,
    kep = 1.00,
    K2t = 5.1,
    K1t = 5.1,
    K3t = 35.0,
    Kkt = 105,
    Ai = 4,
    H = 200.0,

    # Não otimizáveis
    Ad = 6279.0,
    Pcof = 1.0,
    Tuin = 20.0,
    Ebin = 45.0,
    Supin = 1.0,
)
```

Parâmetros ótimos da ONS

In [5]:
```
# Define default parameters
params_ons = dict(
    Ad = 6279.0,
    Str = 100.0,
    K2t = 5.5,
    Crec = 100,
    Ai = 2,
    Capc = 42.0,
    Kkt = 150,
    Pcof = 1.0,
    Tuin = 20.0,
    Ebin = 45.0,
    Supin = 1.0,
    kep = 1.05153505864843,
    H = 200.0,
    K1t = 10.0,
    K3t = 10.0,
)
```

Executar o modelo para intervalo de tempo definido

In [ ]:
```
from modules.smap import ModeloSmapDiario

# start_date = '1995-08-01'
# end_date = '2000-08-01'

start_date = '2000-08-01'
end_date = '2030-01-01'

# Convert DataFrame columns to lists
Ep = df[start_date: end_date]['Ep'].tolist()
Pr = df[start_date: end_date]['Pr'].tolist()

# Call the function with the provided data
result = ModeloSmapDiario(Ep=Ep, Pr=Pr, **params_middle)
```

Salvar resultados do modelo

In [358…
```
# Save result as pandas dataframe
result = pd.DataFrame(result)

# result_df.to_csv('data/optimization/output_base.csv', index=False)
# result_df.to_csv('data/optimization/output_ons.csv', index=False)
result.to_csv('data/optimization/output_base_val.csv', index=False)
# result_df.to_csv('data/optimization/output_ons_val.csv', index=False)

# Print the results
display(result.head(5))
```

	Rsolo	Rsub	Rsup	Rsup2	P	Es	Er	Rec	Ed	Emarg	Ed2	Eb	Q
0	210.000000	94.109247	0.108280	0.0	0.0	0.000000	0.000000	0.0	0.000000	0	0.0	0.000000	0.000000
1	209.130000	93.490040	0.094519	0.0	0.0	0.000000	0.870000	0.0	0.013760	0	0.0	0.619207	46.000000
2	208.821284	92.874907	0.082508	0.0	2.8	0.000000	3.108716	0.0	0.012012	0	0.0	0.615133	45.576835
3	208.791443	92.263822	0.072070	0.0	4.2	0.000048	4.229794	0.0	0.010485	0	0.0	0.611085	45.171768
4	208.005990	91.656758	0.062912	0.0	0.4	0.000000	1.185454	0.0	0.009159	0	0.0	0.607065	44.783173

Calibração - Rotinas de Otimização Automática

Análise gráfica de comportamento das séries geradas (ou simuladas) de vazão face aos valores observados.

Métricas de erro

In [16]:
```python
from modules.metrics import (
    nash_sutcliffe_efficacy,
    relative_error_coefficient,
    correlation_coefficient,
    mean_error,
    normalized_rmse,
    rmse
)

from sklearn.metrics import mean_squared_error
```

Busca de Grade

In [21]:
```python
import time
import numpy as np
import pandas as pd
from sklearn.model_selection import ParameterGrid
from modules.smap import SmapModel

# Define the parameter grid based on the ranges provided
param_grid = {
    'H': np.linspace(0, 200, 5), # 0.96
    'Str': np.linspace(50, 2000, 5), # 1.19
    'K2t': np.linspace(0.2, 10, 5),  # 0.99
    'Crec': np.linspace(0, 100, 5), # 1.00
    'Ai': np.linspace(2, 5, 5), # 0.99
    'Capc': np.linspace(30, 50, 5), # 1.01
    'Kkt': np.linspace(30, 180, 5), # 1.02
    # 'K1t': np.linspace(0.2, 10, 5), # 0.96
    # 'K3t': np.linspace(10, 60, 5), # 0.96
    # 'kep': np.linspace(0.8, 1.2, 5), # 0.96
}

# Convert the parameter grid into a list of dictionaries
param_list = list(ParameterGrid(param_grid))
n_params = len(param_list)

start_date = '1995-08-01'
end_date = '2000-08-01'

data = df[start_date: end_date]
X = data[['Ep', 'Pr']]
y = data['Qobs'].values

# Initialize variables to store the best parameters and best score
best_score = float('inf')
# best_score = - float('inf')
best_params = None
best_result = None

# Example dataframe to hold results
results = []

# Initialize time counter
start = time.time()

# Perform the manual grid search
for i, params in enumerate(param_list):

    # Initialize the model with the current set of parameters
    model = SmapModel(**{
        **params_ons,
        # **params_middle,
        **params,  # Unpack the current parameters from the grid
    })

    # Predict the output
    predictions = model.predict(X)

    # Calculate the score (Mean Squared Error in this case)
    mse = mean_squared_error(y, predictions)
    cef = nash_sutcliffe_efficacy(y, predictions)
    cer = relative_error_coefficient(y, predictions)

    # Collect results
    result = {}
    result['mse'] = mse
    result['cef'] = cef
    result['cer'] = cer
    result['soma_coef'] = cef + cer

    results.append(result)

    # Update the best score and parameters if the current score is better
    if result['mse'] < best_score:
        best_score = result['mse']
        best_params = params
        best_result = result

    time_passed = time.time() - start
    total_time = n_params * time_passed / (i + 1)
    time_left = total_time - time_passed

    time_passed = round(time_passed / 60, 1)
    total_time = round(total_time / 60, 1)
    time_left = round(time_left / 60, 1)

    if i + 1 in range(0, n_params, 100):
        print(f'processing: {i + 1}/{n_params} | {time_passed} m / {total_time} m | {time_left} m', end='\r')

# Convert results to a DataFrame
df_results = pd.DataFrame(results)
```

```python
# Display the best parameters and the best score
print("Best parameters found: ", best_params)
print("Best score: ", best_score)
```

```
Best parameters found:  {'Ai': 2.0, 'Capc': 35.0, 'Crec': 100.0, 'H': 100.0, 'K2t': 10.0, 'Kkt': 30.0, 'Str': 50.0}
Best score:  2887.072405518422
```

Busca Randomizada

In [ ]:
```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from modules.smap import SmapModel
from modules.metrics import nash_sutcliffe_efficacy
from sklearn.metrics import mean_squared_error

def nash_sutcliffe_efficacy_score(estimator, X_test, y_test):
    y_pred = estimator.predict(X_test)
    return nash_sutcliffe_efficacy(y_test, y_pred)

def soma_coef_score(estimator, X_test, y_test):
    y_pred = estimator.predict(X_test)
    cef = nash_sutcliffe_efficacy(y_test, y_pred)
    cer = relative_error_coefficient(y_test, y_pred)
    return - (cef + cer)

start_date = '1995-08-01'
end_date = '2000-08-01'

data = df[start_date: end_date]
X = data[['Ep', 'Pr']]
y = data['Qobs'].values

# Define the parameter distributions (using a wide range with fewer values for random sampling)
param_distributions = {
    'H': uniform(0, 200),
    'Str': uniform(50, 2000),
    'K2t': uniform(0.2, 10),
    'Crec': uniform(0, 100),
    'Ai': uniform(2, 5),
    'Capc': uniform(30, 50),
    'Kkt': uniform(30, 180),
    'K3t': uniform(10, 60),
    'kep': uniform(0.8, 1.2),
}

# Initialize the model
model = SmapModel(**params_ons)

# Perform Randomized Search
random_search = RandomizedSearchCV(model, param_distributions, n_iter=5000, scoring='neg_mean_squared_error', error_score='raise', cv=2, verbose=1)
random_search.fit(X, y)

# Get the best parameters
print(f"Best Parameters: {random_search.best_params_}")
print(f"Best Score: {random_search.best_score_}")
```

Otimização Bayesiana (com as bibliotecas skopt e hyperopt)

In [ ]:
```python
# !pip install scikit-optimize

from skopt import gp_minimize
from skopt.space import Real
from skopt.utils import use_named_args
from modules.smap import SmapModel
from sklearn.metrics import mean_squared_error

start_date = '1995-08-01'
end_date = '2000-08-01'

data = df[start_date: end_date]
X = data[['Ep', 'Pr']]
y = data['Qobs'].values

# Define the search space
search_space = [
    Real(0, 200, name='H'),
    Real(50, 2000, name='Str'),
    Real(0.2, 10, name='K2t'),
    Real(0, 100, name='Crec'),
    Real(2, 5, name='Ai'),
    Real(30, 50, name='Capc'),
    Real(30, 180, name='Kkt'),
    Real(10, 60, name='K3t'),
    Real(0.8, 1.2, name='kep'),
]

# Objective function to minimize
@use_named_args(search_space)
def objective(**params):
    model = SmapModel(**{**params_ons, **params})
    predictions = model.predict(X)
    mse = mean_squared_error(y, predictions)
    return mse

# Perform Bayesian optimization
result = gp_minimize(objective, search_space, n_calls=250, random_state=0, verbose=1)

# Get the best parameters
best_params = {space.name: value for space, value in zip(result.space, result.x)}
print(f"Best Parameters: {best_params}")
print(f"Best Score: {result.fun}")

# Resultados:

# 50: {'H': 199.93598938946454,
#  'Str': 50.0,
#  'K2t': 9.974586382586164,
#  'Crec': 25.19050814571343,
#  'Ai': 5.0,
#  'Capc': 50.0,
#  'Kkt': 30.0,
#  'K3t': 46.33897974946203,
#  'kep': 0.8}

# 250: {'H': 111.99016824287398,
#  'Str': 50.0,
#  'K2t': 9.992900868595116,
#  'Crec': 100.0,
#  'Ai': 2.0,
#  'Capc': 46.762652004145274,
#  'Kkt': 179.53755426503005,
#  'K3t': 10.0,
#  'kep': 1.168066284489991}
```

Algoritmo Genético (usando biblioteca DEAP)

In [ ]:
```python
# !pip install deap
from modules.smap import SmapModel
from deap import base, creator, tools, algorithms
from sklearn.metrics import mean_squared_error
import random

start_date = '1995-08-01'
end_date = '2000-08-01'

data = df[start_date: end_date]
X = data[['Ep', 'Pr']]
y = data['Qobs'].values

# Define the problem as minimization
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# Define the toolbox
toolbox = base.Toolbox()
toolbox.register("H", random.uniform, 0, 200)
toolbox.register("Str", random.uniform, 50, 2000)
toolbox.register("K2t", random.uniform, 0.2, 10)
toolbox.register("Crec", random.uniform, 0, 100)
toolbox.register("Ai", random.uniform, 2, 5)
toolbox.register("Capc", random.uniform, 30, 50)
toolbox.register("Kkt", random.uniform, 30, 180)
toolbox.register("K3t", random.uniform, 10, 60)
toolbox.register("kep", random.uniform, 0.8, 1.2)

# Register individual and population
toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.H, toolbox.Str, toolbox.K2t, toolbox.Crec, toolbox.Ai, toolbox.Capc, toolbox.Kkt, toolbox.K3t, toolbox.kep))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Define the evaluation function
def evaluate(individual):
    params = {
        'H': individual[0],
        'Str': individual[1],
        'K2t': individual[2],
        'Crec': individual[3],
        'Ai': individual[4],
        'Capc': individual[5],
        'Kkt': individual[6],
        'K3t': individual[7],
        'kep': individual[8]
    }
    model = SmapModel(**{**params_ons, **params})
    predictions = model.predict(X)
    try:
        mse = mean_squared_error(y, predictions)
    except:
        print(y)
        print(predictions)
        raise
    return (mse,)

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)

# Perform the genetic algorithm
population = toolbox.population(n=100)
algorithms.eaSimple(population, toolbox, cxpb=0.7, mutpb=0.2, ngen=150, verbose=True)

# Get the best individual
individual = tools.selBest(population, 1)[0]
best_params = {
    'H': individual[0],
    'Str': individual[1],
    'K2t': individual[2],
    'Crec': individual[3],
    'Ai': individual[4],
    'Capc': individual[5],
    'Kkt': individual[6],
    'K3t': individual[7],
    'kep': individual[8]
}
print(f"Best parameters:")
display(best_params)
```

Gerando valores de vazão com parametros calibrados

In [28]:
```python
from modules.smap import ModeloSmapDiario

params = {
    **params_ons,
    **{
        'H': 78.78471407043702,
        'Str': 191.04762937207383,
        'K2t': 8.087543050695219,
        'Crec': 94.99793281367697,
        'Ai': -9.903240185866013,
        'Capc': 49.724993099393416,
        'Kkt': 90.1798512276936,
        'K3t': 9.948916908314086,
        'kep': -0.057373672702039094
    }
}

start_date = '1995-08-01'
end_date = '2000-08-01'

start_date = '2000-08-01'
end_date = '2030-01-01'

# Convert DataFrame columns to lists
Ep = df[start_date: end_date]['Ep'].tolist()
Pr = df[start_date: end_date]['Pr'].tolist()

# Call the function with the provided data
result = ModeloSmapDiario(
    Ep=Ep,
    Pr=Pr,
    **{**params_ons, **params}
)
```

Salvando valores de vazão gerados

In [29]:
```python
# Save result as pandas dataframe
result_df = pd.DataFrame(result)

# result_df.to_csv('data/optimization/output_opt_grid.csv', index=False)
# result_df.to_csv('data/optimization/output_opt_randomized.csv', index=False)
# result_df.to_csv('data/optimization/output_opt_bayesian.csv', index=False)
# result_df.to_csv('data/optimization/output_opt_genetic_250.csv', index=False)

# result_df.to_csv('data/optimization/output_opt_grid_val.csv', index=False)
# result_df.to_csv('data/optimization/output_opt_randomized_val.csv', index=False)
# result_df.to_csv('data/optimization/output_opt_bayesian_val.csv', index=False)
result_df.to_csv('data/optimization/output_opt_genetic_val_250.csv', index=False)

# Print the results
display(result_df.head(5))
```

	Rsolo	Rsub	Rsup	Rsup2	P	Es	Er	Rec	Ed	Emarg	Ed2	Eb	Q
0	38.209526	80.870069	0.167530	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.000000
1	37.218998	0.000000	0.756408	0.0	0.0	0.602638	0.387889	0.0	0.013760	0.0	0.0	0.619207	46.000000
2	34.699981	0.000000	1.663298	0.0	2.8	0.969018	4.350000	0.0	0.062128	0.0	0.0	0.000000	4.515064
3	33.383067	0.000000	2.693595	0.0	4.2	1.166913	4.350000	0.0	0.136616	0.0	0.0	0.000000	9.928365
4	28.801061	0.000000	3.104362	0.0	0.4	0.632007	4.350000	0.0	0.221240	0.0	0.0	0.000000	16.078298

Análise gráfica de comportamento das séries geradas

Fonte: http://www.coc.ufrj.br/pt/dissertacoes-de-mestrado/105-msc-pt-2005/1992-rafael-carneiro-di-bello

(a) Coeficiente de correlação (R)

$$R = \frac{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - m_x)(x_i^* - m_x^*)}{\sigma_x \sigma_x^*}$$

onde

x_i é um valor observado;

x_i^* é um valor simulado;

m_x é a média dos valores observados;

m_x^* é a média dos valores simulados;

σ_x é o desvio padrão dos valores observados;

σ_x^* é o desvio padrão dos valores simulados; e

N é o número de medidas (dias simulados).

(b) Erro médio (EM)

$$EM = \frac{1}{N}\sum_{i=1}^{N}(x_i - x_i^*)$$

(c) Erro reduzido médio quadrático (ERM)

$$ERM = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i - x_i^*}{\sigma_x}\right)^2$$

166

(d) Erro médio quadrático (EMQ)

$$EMQ = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - x_i^*)^2}$$

Curvas de permanência

A "curva de permanência" é um gráfico que mostra a probabilidade de uma determinada vazão ser igualada ou excedida.

Gráfico de dispersão

Este tipo de gráfico é útil para analisar a correlação entre as vazões observadas e as geradas.

- **Gráfico de Dispersão**:
  - O gráfico plota pontos para cada par de valores de vazão observada e gerada.
  - As previsões dos diferentes modelos são diferenciadas por cor e legendadas.
- **Linha de Perfeição**:
  - A linha preta tracejada (`Linha de Perfeição`) representa onde os valores gerados seriam exatamente iguais aos valores observados. Isso ajuda a visualizar o quão próximo as previsões estão dos valores reais.

Esse gráfico permite avaliar visualmente o desempenho dos modelos de previsão em relação aos dados observados.

Gráfico de Resíduos

Mostra a diferença entre as vazões geradas e as vazões observadas (resíduos) ao longo do tempo.

Esse tipo de gráfico é útil para identificar padrões ou desvios sistemáticos nos erros de previsão.

Recarregar valores de vazão gerados

In [6]:
```python
import pandas as pd
import numpy as np

output_base = pd.read_csv('data/optimization/output_base.csv')
output_ons = pd.read_csv('data/optimization/output_ons.csv')
output_opt_grid = pd.read_csv('data/optimization/output_opt_grid.csv')
output_opt_bay = pd.read_csv('data/optimization/output_opt_bayesian.csv')
output_opt_rand = pd.read_csv('data/optimization/output_opt_randomized.csv')
output_opt_gen = pd.read_csv('data/optimization/output_opt_genetic_250.csv')
```

```python
output_base.columns += ' - BASE'
output_ons.columns += ' - ONS'
output_opt_grid.columns += ' - OPT GRID'
output_opt_rand.columns += ' - OPT RAND'
output_opt_bay.columns += ' - OPT BAY'
output_opt_gen.columns += ' - OPT GEN'

preds = pd.concat([output_base, output_ons, output_opt_grid, output_opt_rand, output_opt_bay, output_opt_gen], axis=1)

start_date = '1995-08-01'
end_date = '2000-08-01'
data = df[start_date: end_date]

preds.index = data.index
preds.index.name = 'index'
preds['Q - OBS'] = data['Qobs']

preds['Q - OPT'] = preds['Q - OPT GEN']
preds['Rsub - OPT'] = preds['Rsub - OPT GEN']
preds['Rsup - OPT'] = preds['Rsup - OPT GEN']

preds = pd.concat([preds, df.loc[preds.index]], axis=1)

preds.head()
```

Out[6]:

index	Rsolo - BASE	Rsub - BASE	Rsup - BASE	Rsup2 - BASE	P - BASE	Es - BASE	Er - BASE	Rec - BASE	Ed - BASE	Emarg - BASE	...	Ed2 - OPT GEN	Eb - OPT GEN	Q - OPT GEN	Q - OBS	Q - OPT	Rsub - OPT	Rsup - OPT	Qobs	Ep	Pr
1995-08-01	210.000000	94.109247	0.108280	0.0	0.0	0.0	0.000000	0.0	0.000000	0	...	0.0	0.000000	0.000000	46	0.000000	80.870069	0.167530	46	4.35	0.0
1995-08-02	209.130000	93.490040	0.094519	0.0	0.0	0.0	0.870000	0.0	0.013760	0	...	0.0	0.619207	46.000000	46	46.000000	0.000000	0.756408	46	4.35	0.0
1995-08-03	208.263604	92.874907	0.082508	0.0	0.0	0.0	0.866396	0.0	0.012012	0	...	0.0	0.000000	4.515064	46	4.515064	0.000000	1.293273	46	4.35	0.0
1995-08-04	207.400798	92.263822	0.072023	0.0	0.0	0.0	0.862806	0.0	0.010485	0	...	0.0	0.000000	7.719654	46	7.719654	0.000000	1.782535	46	4.35	0.0
1995-08-05	206.541566	91.656758	0.062870	0.0	0.0	0.0	0.859232	0.0	0.009153	0	...	0.0	0.000000	10.640103	46	10.640103	0.000000	2.228237	46	4.35	0.0

5 rows × 85 columns

Calcular métricas de erro - Calibração

In [6]:

```python
y_true = preds['Q - OBS']
columns = ['Q - BASE', 'Q - ONS', 'Q - OPT GRID', 'Q - OPT RAND', 'Q - OPT BAY', 'Q - OPT GEN']

stats = []
for pred in columns:
    y_pred = preds[pred]

    # Calculate the score (Mean Squared Error in this case)
    mse = mean_squared_error(y_true, y_pred)
    cef = nash_sutcliffe_efficacy(y_true, y_pred)
    cer = relative_error_coefficient(y_true, y_pred)
    soma_coef = cef + cer

    cc = correlation_coefficient(y_true, y_pred)
    me = mean_error(y_true, y_pred)
    rmse_norm = normalized_rmse(y_true, y_pred)
    RMSE = rmse(y_true, y_pred)

    stats.append({
        'cef': cef,
        'cer': cer,
        'soma_coef': soma_coef,
        'cc': cc,
        'me': me,
        'rmse_norm': rmse_norm,
        'rmse': RMSE}
    )

stats = pd.DataFrame(stats, index=columns)
display(stats.T)
```

	Q - BASE	Q - ONS	Q - OPT GRID	Q - OPT RAND	Q - OPT BAY	Q - OPT GEN
cef	0.316289	0.793380	0.627835	0.482649	0.596397	0.851605
cer	0.651888	0.871702	0.669828	0.666273	0.617783	0.811392
soma_coef	0.968177	1.665082	1.297663	1.148922	1.214180	1.662998
cc	0.634245	0.904831	0.869941	0.888134	0.882859	0.923651
me	25.853598	-0.028345	10.806848	-2.055290	15.456251	0.528001
rmse_norm	0.683337	0.206507	0.371962	0.517068	0.403382	0.148314
rmse	72.827808	40.035637	53.731484	63.351000	55.954900	33.928933

Comparando valores de vazão gerados e observados

In [7]:

```python
from modules.visu import report, interactive_report

# report(preds)
figures = interactive_report(preds)
```

Comparação das Séries ao Longo do Tempo

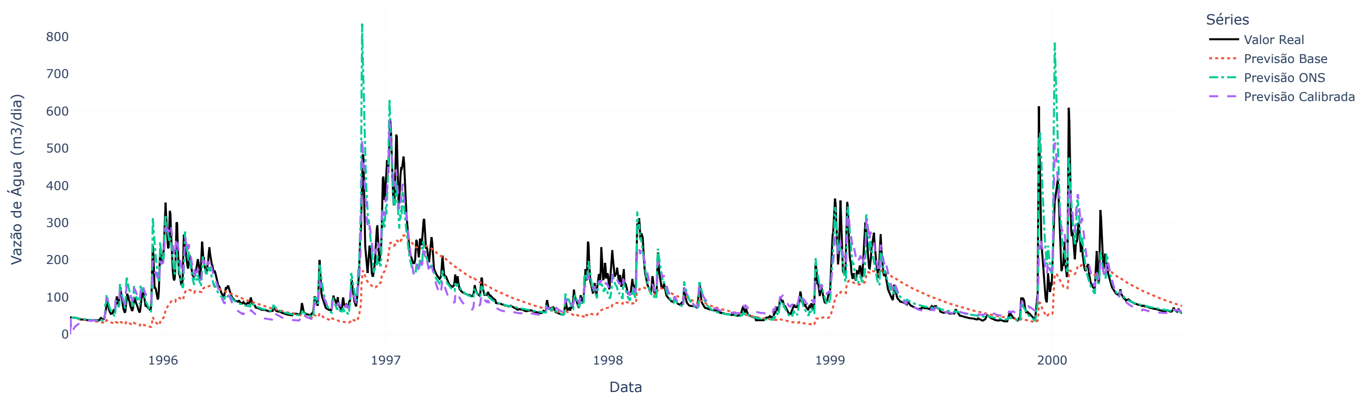
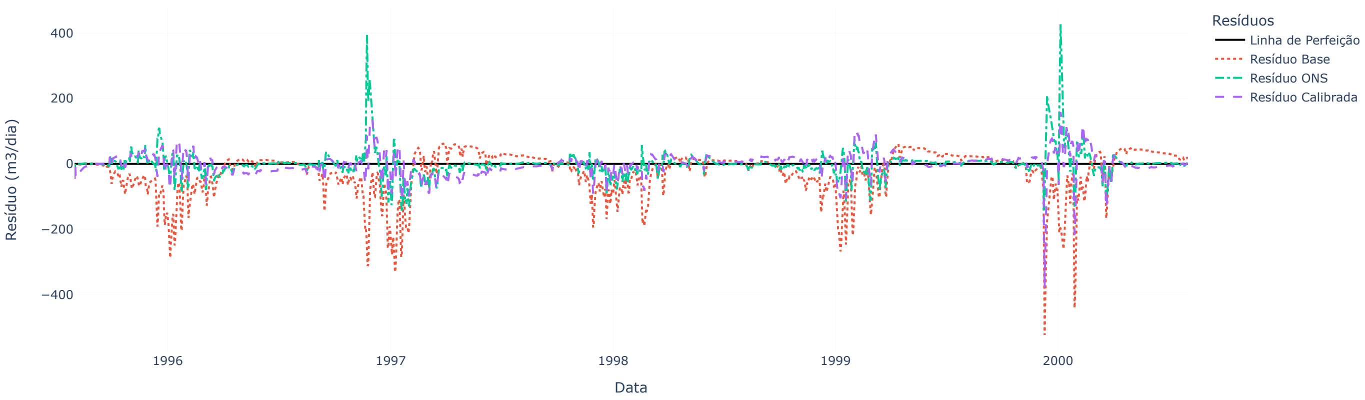


Gráfico de Resíduos



Comparação das Séries Acumuladas ao Longo do Tempo

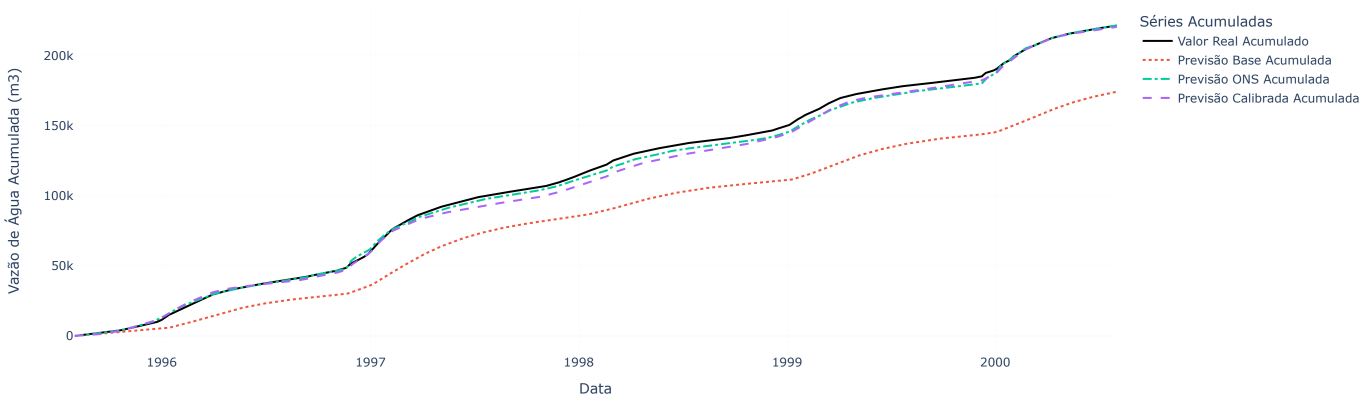
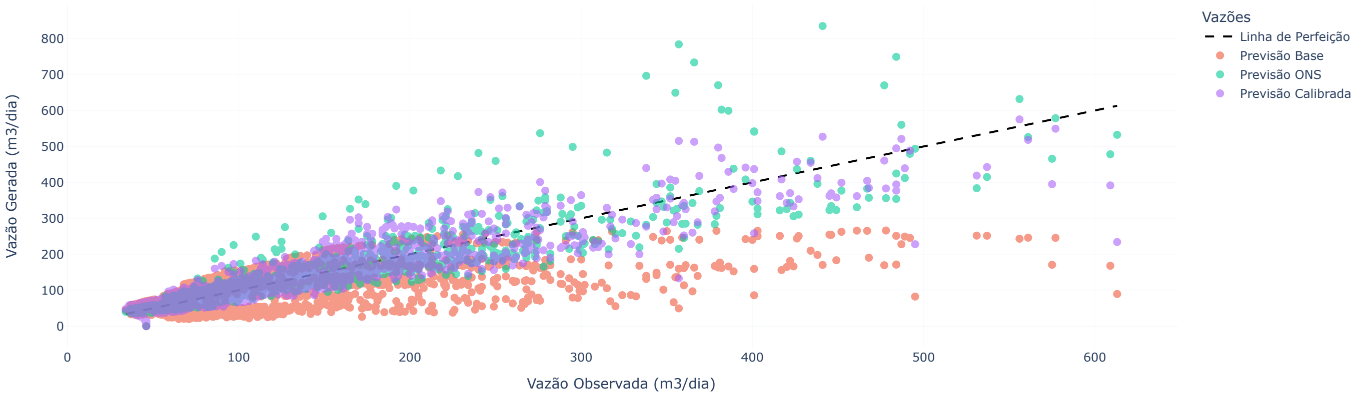
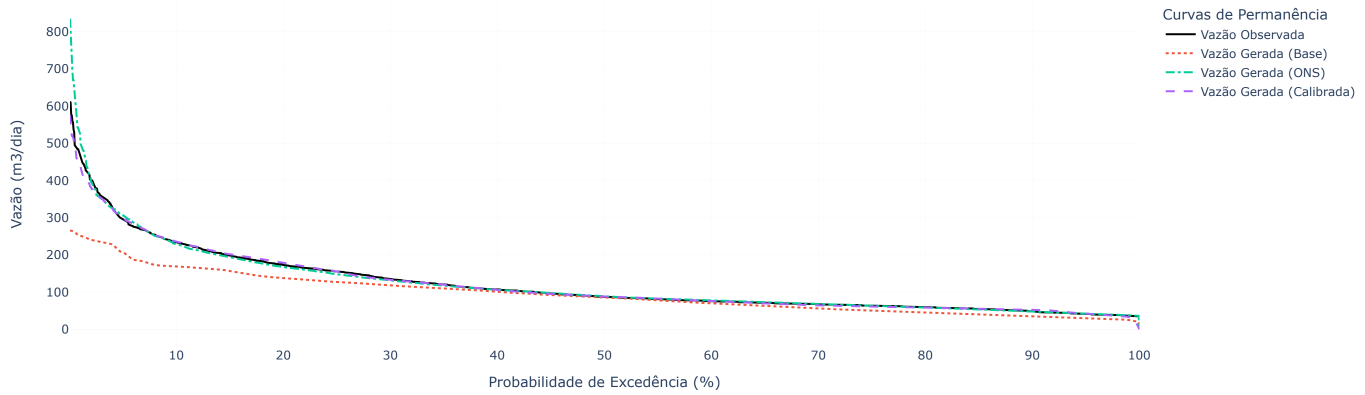


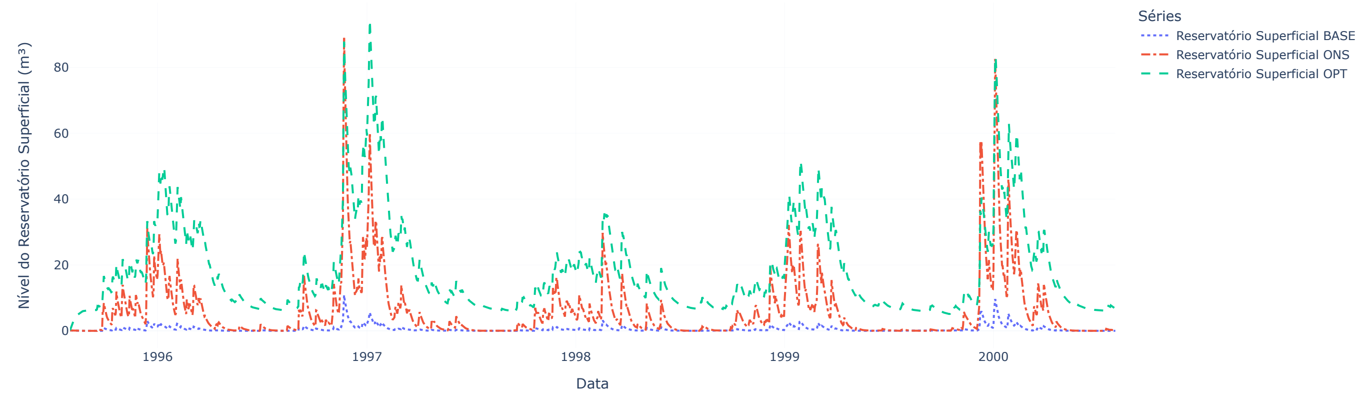
Gráfico de Dispersão: Vazão Gerada vs Vazão Observada



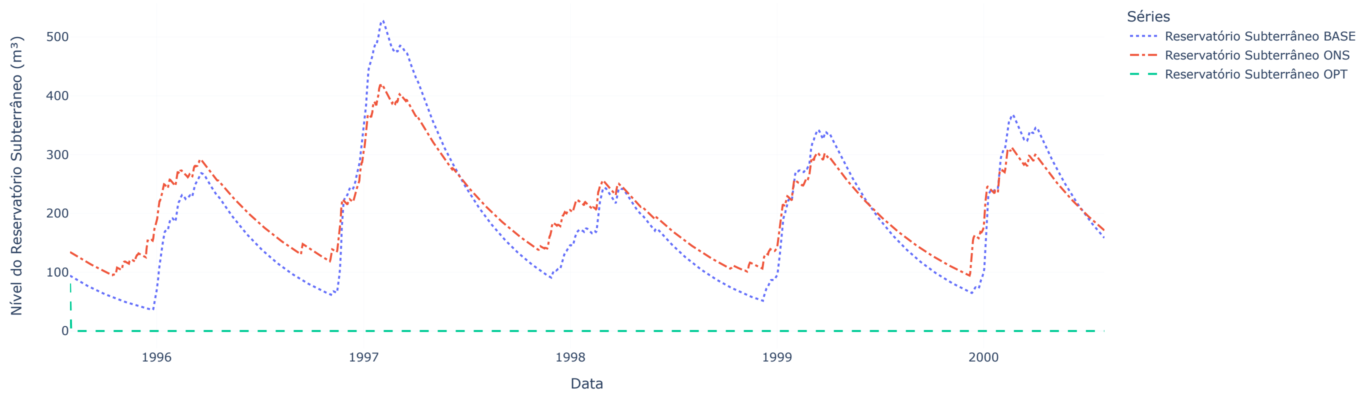
Curva de Permanência das Vazões



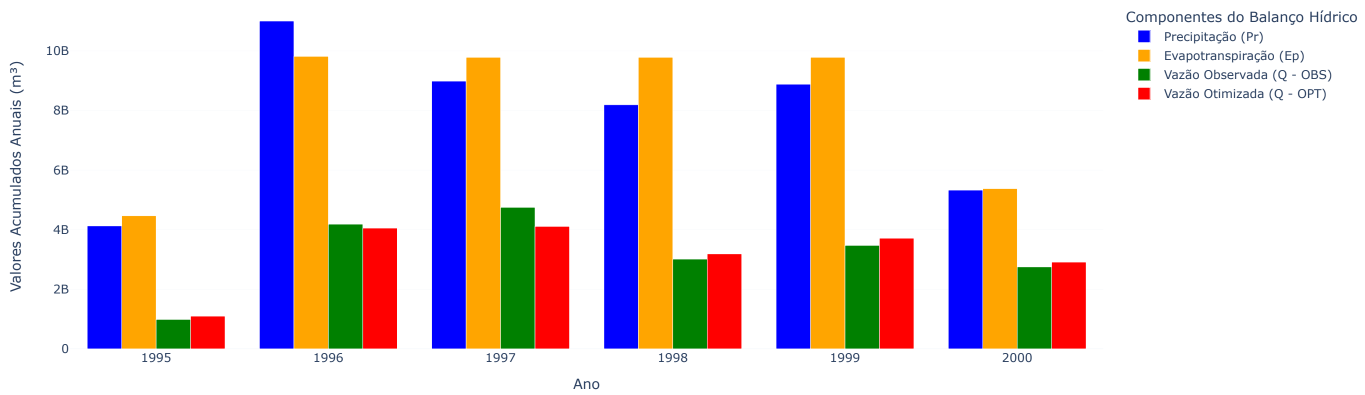
Variação dos Níveis do Reservatório Superficial



Variação dos Níveis do Reservatório Subterrâneo



Balanço Hídrico Anual (m³/ano)



Salvando gráficos como html

In [8]:
```
import plotly.graph_objects as go
import plotly.io as pio

# Combine figures into a single HTML file
with open('camargos_calibracao.html', 'w') as out:
    for fig in figures:
        out.write(fig.to_html(full_html=False, include_plotlyjs='cdn'))
```

Periodo de Validação

Recarregar valores de vazão gerados

In [8]:
```
import pandas as pd
import numpy as np

output_base = pd.read_csv('data/optimization/output_base_val.csv')
output_ons = pd.read_csv('data/optimization/output_ons_val.csv')
output_opt_rand = pd.read_csv('data/optimization/output_opt_randomized_val.csv')
output_opt_gen = pd.read_csv('data/optimization/output_opt_genetic_val_250.csv')

output_base.columns += ' - BASE'
output_ons.columns += ' - ONS'
output_opt_rand.columns += ' - OPT RAND'
output_opt_gen.columns += ' - OPT GEN'

preds = pd.concat([output_base, output_ons, output_opt_rand, output_opt_gen], axis=1)

# start_date = '1995-08-01'
# end_date = '2000-08-01'
start_date = '2000-08-01'
end_date = '2030-01-01'
data = df[start_date: end_date]

preds.index = data.index
preds.index.name = 'index'
preds['Q - OBS'] = data['Qobs']

preds['Q - OPT'] = preds['Q - OPT GEN']
preds['Rsub - OPT'] = preds['Rsub - OPT GEN']
preds['Rsup - OPT'] = preds['Rsup - OPT GEN']

preds = pd.concat([preds, df.loc[preds.index]], axis=1)

preds.head()
```

Out[8]:

index	Rsolo - BASE	Rsub - BASE	Rsup - BASE	Rsup2 - BASE	P - BASE	Es - BASE	Er - BASE	Rec - BASE	Ed - BASE	Emarg - BASE	...	Ed2 - OPT GEN	Eb - OPT GEN	Q - OPT GEN	Q - OBS	Q - OPT	Rsub - OPT	Rsup - OPT	Qobs	Ep	Pr
2000-08-01	210.000000	94.109247	0.108280	0.0	0.0	0.000000	0.000000	0.0	0.000000	0	...	0.0	0.000000	0.000000	56	0.000000	80.870069	0.167530	56	4.35	0.0
2000-08-02	209.130000	93.490040	0.094519	0.0	0.0	0.000000	0.870000	0.0	0.013760	0	...	0.0	0.619207	46.000000	56	46.000000	0.000000	0.756408	56	4.35	0.0
2000-08-03	208.821284	92.874907	0.082508	0.0	2.8	0.000000	3.108716	0.0	0.012012	0	...	0.0	0.000000	4.515064	56	4.515064	0.000000	1.663298	56	4.35	2.8
2000-08-04	208.791443	92.263822	0.072070	0.0	4.2	0.000048	4.229794	0.0	0.010485	0	...	0.0	0.000000	9.928365	58	9.928365	0.000000	2.693595	58	4.35	4.2
2000-08-05	208.005990	91.656758	0.062912	0.0	0.4	0.000000	1.185454	0.0	0.009159	0	...	0.0	0.000000	16.078298	62	16.078298	0.000000	3.104362	62	4.35	0.4

5 rows × 59 columns

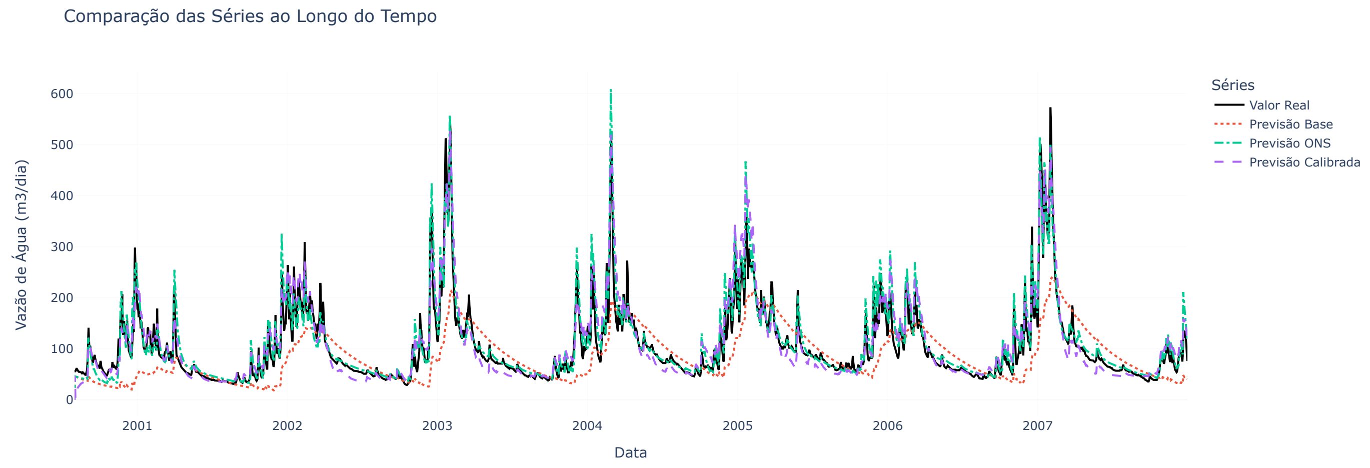
Calcular métricas de erro - Periodo de Validação

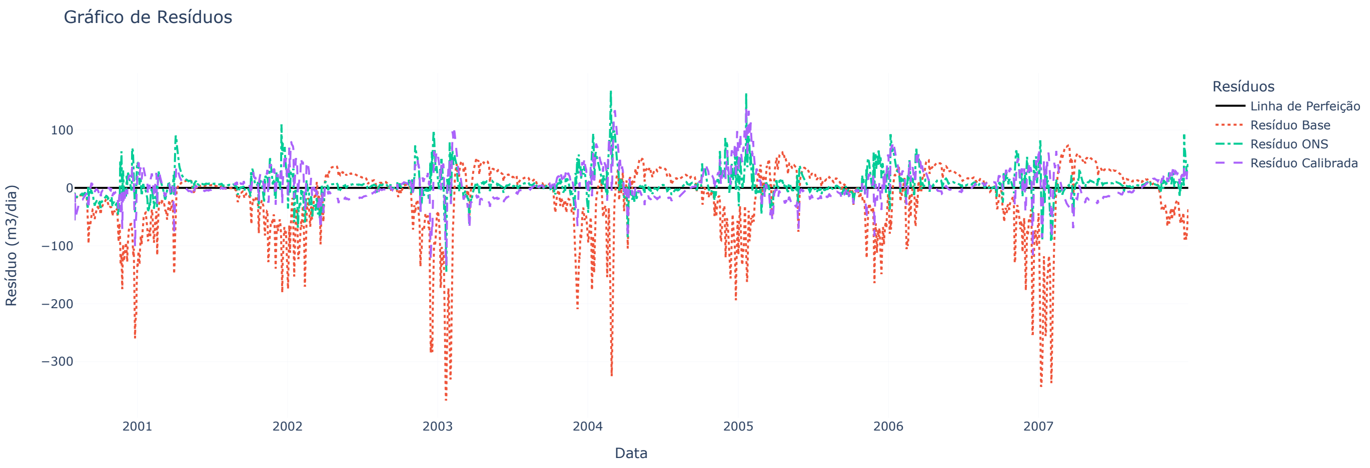
In [11]:
```
y_true = preds['Q - OBS']
columns = ['Q - BASE', 'Q - ONS', 'Q - OPT RAND', 'Q - OPT GEN']

stats = []
for pred in columns:
    y_pred = preds[pred]

    # Calculate the score (Mean Squared Error in this case)
    mse = mean_squared_error(y_true, y_pred)
    cef = nash_sutcliffe_efficacy(y_true, y_pred)
    cer = relative_error_coefficient(y_true, y_pred)
    soma_coef = cef + cer

    cc = correlation_coefficient(y_true, y_pred)
    me = mean_error(y_true, y_pred)
    rmse_norm = normalized_rmse(y_true, y_pred)
    RMSE = rmse(y_true, y_pred)

    stats.append({
        'cef': cef,
        'cer': cer,
        'soma_coef': soma_coef,
        'cc': cc,
        'me': me,
        'rmse_norm': rmse_norm,
        'rmse': RMSE}
    )

stats = pd.DataFrame(stats, index=columns)
display(stats.T)
```

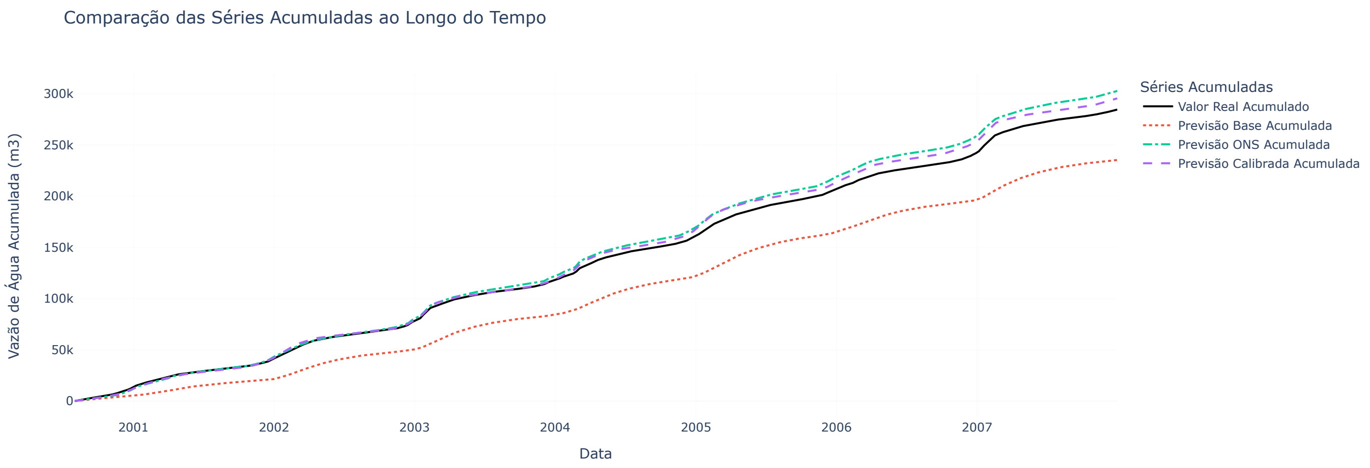
	Q - BASE	Q - ONS	Q - OPT RAND	Q - OPT GEN
cef	0.296795	0.898135	0.453340	0.854966
cer	0.649947	0.856959	0.562850	0.807230
soma_coef	0.946741	1.755094	1.016191	1.662195
cc	0.601270	0.957205	0.917602	0.940256
me	18.175795	-6.738186	0.969686	-4.042630
rmse_norm	0.702946	0.101828	0.546458	0.144981
rmse	62.102310	23.636318	54.755173	28.203450

Comparação dos valores de vazão gerados e observados

In [9]:
```python
from modules.visu import interactive_report

# report(preds)
figures_val = interactive_report(preds)
```







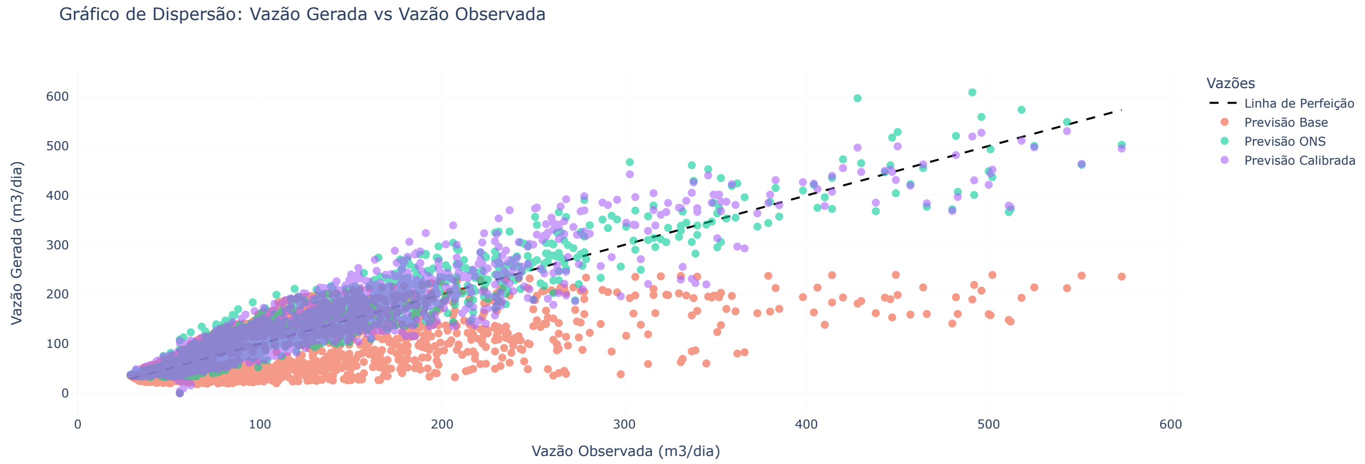
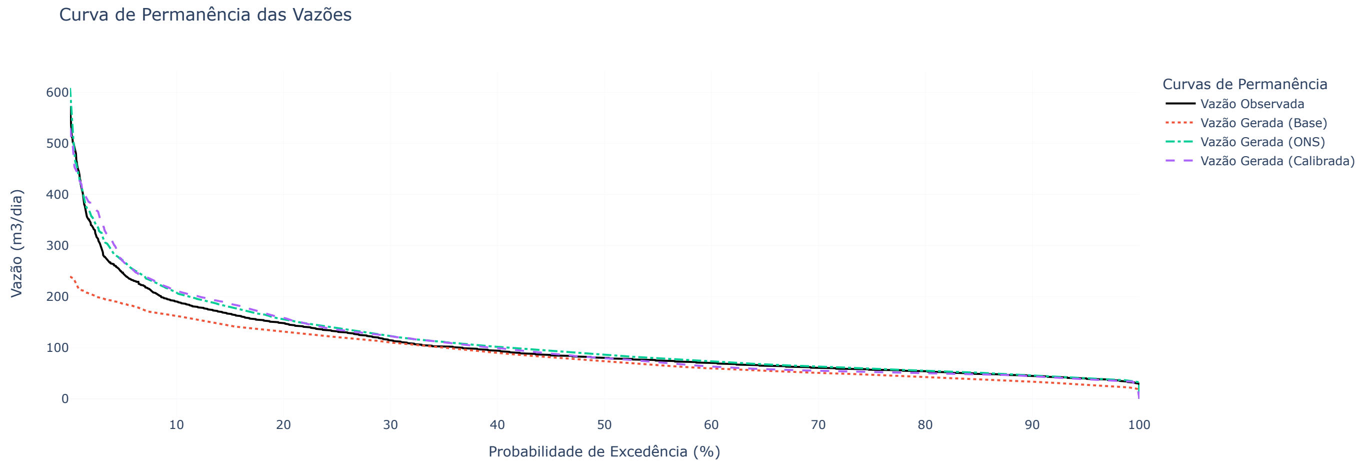
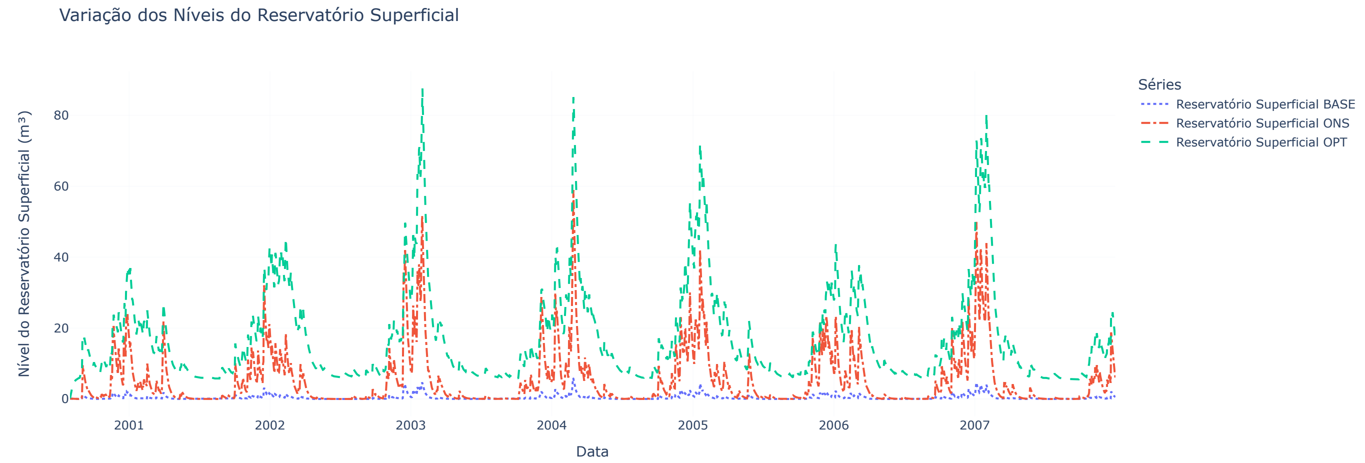
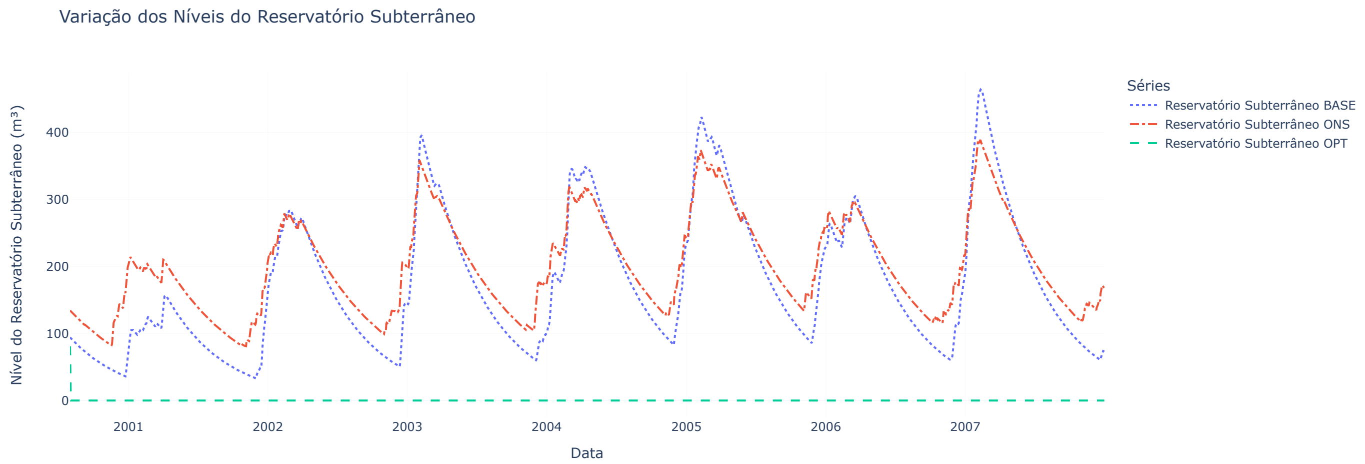
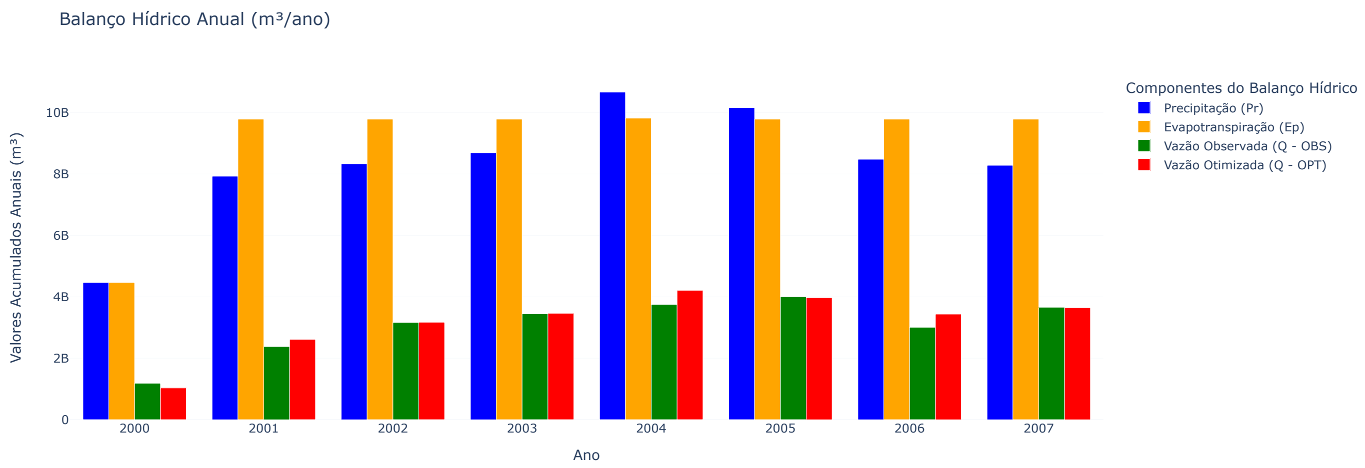


Gráfico de Dispersão: Vazão Gerada vs Vazão Observada
Vazão Gerada (m3/dia)
Vazão Observada (m3/dia)
Vazões
Linha de Perfeição
Previsão Base
Previsão ONS
Previsão Calibrada


Curva de Permanência das Vazões
Vazão (m3/dia)
Probabilidade de Excedência (%)
Curvas de Permanência
Vazão Observada
Vazão Gerada (Base)
Vazão Gerada (ONS)
Vazão Gerada (Calibrada)


Variação dos Níveis do Reservatório Superficial
Nível do Reservatório Superficial (m³)
Data
Séries
Reservatório Superficial BASE
Reservatório Superficial ONS
Reservatório Superficial OPT


Variação dos Níveis do Reservatório Subterrâneo
Nível do Reservatório Subterrâneo (m³)
Data
Séries
Reservatório Subterrâneo BASE
Reservatório Subterrâneo ONS
Reservatório Subterrâneo OPT



Salvando gráficos como html

In [13]:
```
import plotly.graph_objects as go
import plotly.io as pio

# Combine figures into a single HTML file
with open('camargos.html', 'w', encoding='utf-8') as out:
    out.write('''<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Análise dos Resultados do Modelo SMAP para a Bacia de Camargos</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <style>
        body {font-family: 'Arial', sans-serif; margin: 0; padding: 0; background-color: #f4f4f4; color: #333;}
        .container {width: 90%; max-width: 1200px; margin: 20px auto; padding: 20px; background-color: #fff; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); border-radius: 8px;}
        h1, h2 {text-align: center; color: #2c3e50;}
        h1 {margin-bottom: 30px;}
        h2 {margin-bottom: 20px; margin-top: 40px;}
        .section {margin-bottom: 40px;}
        .plot-container {text-align: center;}
        footer {text-align: center; margin-top: 30px; padding: 10px 0; background-color: #2c3e50; color: white;}
    </style>
</head>
<body>
<div class="container">
<h1>Análise dos Resultados do Modelo SMAP para a Bacia de Camargos</h1>
<div class="section">
<h2>Calibração do Modelo (Ago 1995 a Jul 2000)</h2>
<div class="plot-container">''')

    for fig in figures:
        out.write(fig.to_html(full_html=False, include_plotlyjs='cdn'))

    out.write('''</div></div><div class="section"><h2>Validação do Modelo (Ago 2000 a Dez 2007)</h2><div class="plot-container">''')

    for fig in figures_val:
        out.write(fig.to_html(full_html=False, include_plotlyjs='cdn'))

    out.write('''</div></div></div><footer>&copy; 2024 Análise dos Resultados do Modelo SMAP - Bacia de Camargos</footer></body></html>''')
```