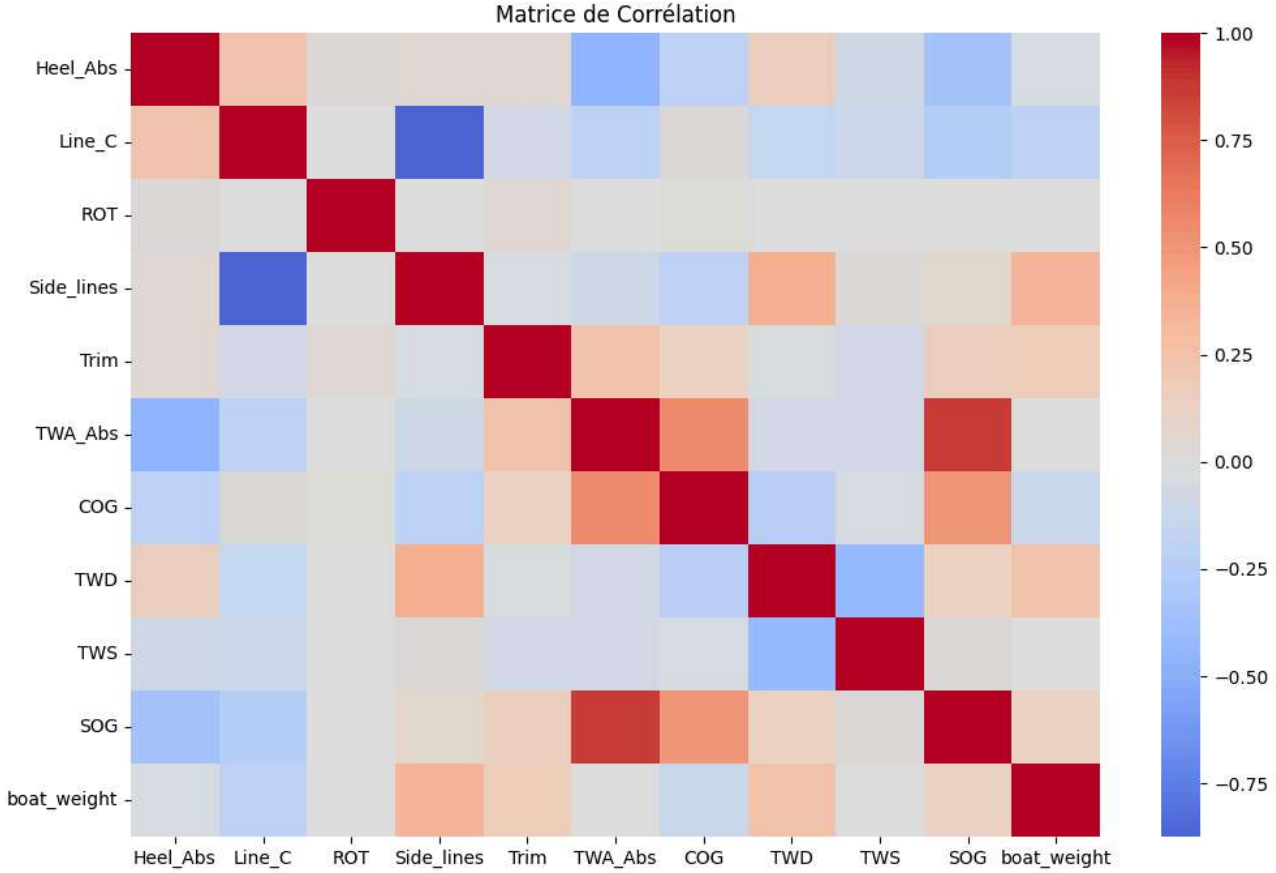
In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from statsmodels.stats.outliers_influence import variance_inflation_factor

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
import statsmodels.formula.api as smf
import statsmodels.api as sm

import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures, StandardScaler, FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

import warnings
warnings.filterwarnings("ignore")

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("all_data.csv")
```

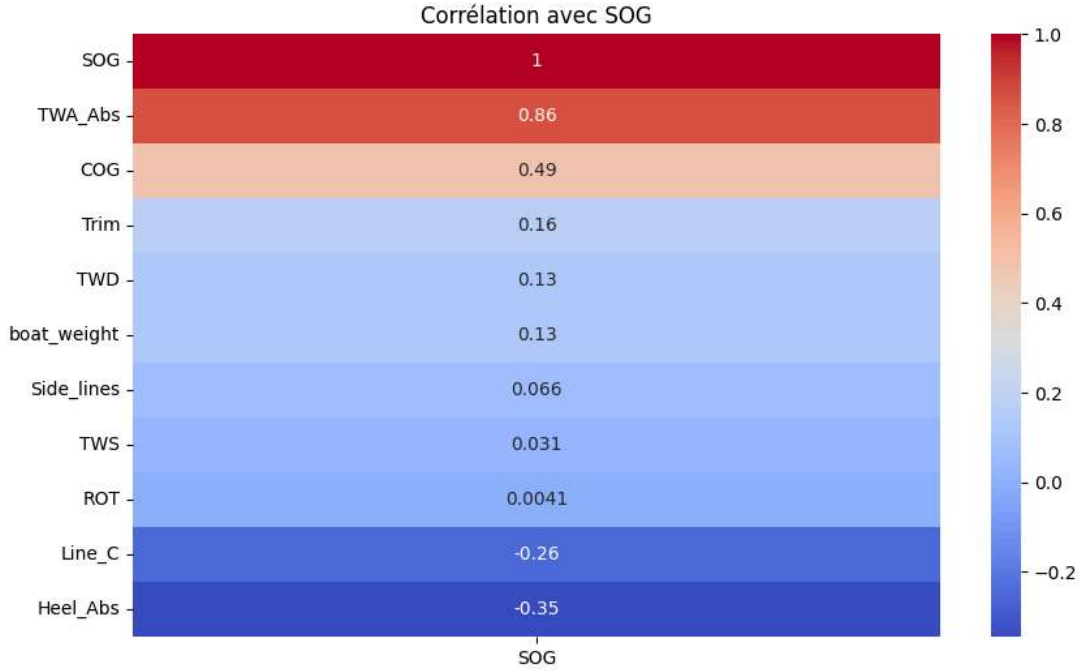
In [2]:
```
def afficher_matrice_correlation(dataframe, taille_figure=(12, 8), cmap="coolwarm"):
    plt.figure(figsize=taille_figure)
    sns.heatmap(dataframe.corr(), cmap=cmap, center=0)
    plt.title("Matrice de Corrélation")
    plt.show()

df_numeric = df.select_dtypes(include=["float64", "int64"]).copy()
drop_cols = [
    "TWA", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE","Total_lines","VMG","gain_forward","gain_lateral", "gain_vmg"
]
df_numeric.drop(columns=[c for c in drop_cols if c in df_numeric.columns], inplace=True)
df_numeric.dropna(subset=["SOG"], inplace=True)
print(f"Variables utilisées:", df_numeric.columns.tolist())
afficher_matrice_correlation(df_numeric)
```

```
Variables utilisées: ['Heel_Abs', 'Line_C', 'ROT', 'Side_lines', 'Trim', 'TWA_Abs', 'COG', 'TWD', 'TWS', 'SOG', 'boat_weight']
```



In [3]:
```python
def afficher_correlation_vmg(df, variable="SOG", taille_figure=(10, 6)):
    corr_with_vmg = df.corr()[variable].sort_values(ascending=False)
    print(f"Corrélation avec {variable} :")
    print(corr_with_vmg)

    plt.figure(figsize=taille_figure)
    sns.heatmap(corr_with_vmg.to_frame(), annot=True, cmap="coolwarm")
    plt.title(f"Corrélation avec {variable}")
    plt.show()

afficher_correlation_vmg(df_numeric, variable="SOG")
```

```
Corrélation avec SOG :
SOG            1.000000
TWA_Abs        0.862650
COG            0.489489
Trim           0.160931
TWD            0.129801
boat_weight    0.128466
Side_lines     0.065668
TWS            0.031173
ROT            0.004052
Line_C        -0.255391
Heel_Abs      -0.345028
Name: SOG, dtype: float64
```



In [4]:
```
"""
def calculer_anova_quadratique(df, target="SOG"):
    cols = df.columns.drop(target)
    formula = f"{target} ~ " + " + ".join(
        list(cols) +
        [f"I({col}**2)" for col in cols] +
        [f"I({col}**(1/2))" for col in cols]
    )
    model_anova = smf.ols(formula=formula, data=df.dropna(subset=cols)).fit()
    anova_table = sm.stats.anova_lm(model_anova, typ=2)
    return anova_table.sort_values("F", ascending=False)
"""



def calculer_anova_quadratique(df, target="SOG"):
    # Make a copy of the DataFrame to avoid modifying the original
```

```
    df_copy = df.copy()

    cols = df_copy.columns.drop(target)

    # Print missing value count before imputation
    print("Missing values before imputation:\n", df_copy[cols].isna().sum())

    # Impute missing values with column means
    df_copy[cols] = df_copy[cols].apply(lambda x: x.fillna(x.mean()))

    # Print shape after imputation
    print("Shape after imputation:", df_copy[cols].shape)

    # Build the formula with linear, squared, and square root terms
    formula = f"{target} ~ " + " + ".join(
        list(cols) +
        [f"I({col}**2)" for col in cols] +
        [f"I({col}**(1/2))" for col in cols]
    )

    # Fit the model
    model_anova = smf.ols(formula=formula, data=df_copy).fit()
    anova_table = sm.stats.anova_lm(model_anova, typ=2)

    return anova_table.sort_values("F", ascending=False)

resultats_anova = calculer_anova_quadratique(df_numeric)
resultats_anova
```

```
Missing values before imputation:
 Heel_Abs          0
Line_C          763
ROT               0
Side_lines     9445
Trim              0
TWA_Abs           0
COG               0
TWD               0
TWS               0
boat_weight       0
dtype: int64
Shape after imputation: (80131, 10)
```

Out[4]:

	sum_sq	df	F	PR(>F)
I(TWA_Abs ** (1 / 2))	1088.761102	1.0	1382.226999	1.686478e-297
TWA_Abs	1081.785978	1.0	1373.371792	1.223524e-295
I(Side_lines ** (1 / 2))	923.814589	1.0	1172.820617	2.166319e-253
Side_lines	795.733582	1.0	1010.216510	5.555243e-219
I(boat_weight ** 2)	680.762482	1.0	864.255971	5.649139e-188
boat_weight	669.549502	1.0	850.020631	6.057039e-185
I(boat_weight ** (1 / 2))	663.787861	1.0	842.705991	2.186551e-183
I(TWA_Abs ** 2)	636.698332	1.0	808.314719	4.640805e-176
I(Side_lines ** 2)	550.984068	1.0	699.496936	7.810476e-153
I(COG ** 2)	148.121549	1.0	188.046399	1.058772e-42
I(Line_C ** 2)	61.553448	1.0	78.144633	9.955580e-19
TWS	47.677951	1.0	60.529120	7.421912e-15
I(TWS ** (1 / 2))	47.299985	1.0	60.049277	9.467372e-15
I(TWD ** 2)	36.699165	1.0	46.591099	8.869273e-12
I(TWS ** 2)	35.989243	1.0	45.689824	1.404248e-11
I(Heel_Abs ** 2)	29.943714	1.0	38.014776	7.087612e-10
TWD	26.036774	1.0	33.054755	9.024555e-09
I(TWD ** (1 / 2))	20.051148	1.0	25.455757	4.546097e-07
Line_C	18.524706	1.0	23.517876	1.242196e-06
Heel_Abs	14.759847	1.0	18.738233	1.503071e-05
COG	9.473944	1.0	12.027561	5.247433e-04
I(COG ** (1 / 2))	9.321881	1.0	11.834511	5.820185e-04
I(Heel_Abs ** (1 / 2))	7.427948	1.0	9.430086	2.135965e-03
I(Trim ** 2)	3.182033	1.0	4.039721	4.444782e-02
I(ROT ** 2)	1.407513	1.0	1.786896	1.813123e-01
I(ROT ** (1 / 2))	1.351290	1.0	1.715518	1.902784e-01
ROT	1.159531	1.0	1.472072	2.250262e-01
I(Line_C ** (1 / 2))	0.406640	1.0	0.516246	4.724515e-01
Trim	0.384540	1.0	0.488190	4.847400e-01
I(Trim ** (1 / 2))	0.185713	1.0	0.235771	6.272801e-01
Residual	31727.211565	40279.0	NaN	NaN

In [5]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.impute import SimpleImputer

def analyser_modele_polynomial_final(df, target="SOG", degree=2, top_coefs=30):
    """
    Version finale qui gère:
    - Les NaN via imputation
    - Les racines carrées sans redondance
    - Les noms de variables clairs
    """
    cols = df.columns.drop(target)

    X = df[cols].copy()
    X_sqrt = X.apply(lambda x: np.sqrt(np.abs(x))).add_prefix('sqrt_')
    X_combined = pd.concat([X, X_sqrt], axis=1)
    y = df[target].copy()

    imputer = SimpleImputer(strategy='mean')
    X_imputed = pd.DataFrame(imputer.fit_transform(X_combined),
                             columns=X_combined.columns)

    if y.isna().any():
        y = y.fillna(y.mean())

    model = make_pipeline(
        StandardScaler(),
        LinearRegression()
    )

    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_poly = poly.fit_transform(X_imputed)
    feature_names = poly.get_feature_names_out(X_combined.columns)

    model.fit(X_poly, y)
    y_pred = model.predict(X_poly)
    print(f"R²: {r2_score(y, y_pred):.3f}")
    print(f"Échantillons utilisés: {len(X_imputed)}")

    coef_df = pd.DataFrame({
        'feature': feature_names,
        'coefficient': model.named_steps['linearregression'].coef_
    }).sort_values('coefficient', key=abs, ascending=False)

    def is_valid_feature(name):
        if 'sqrt_' in name:
            base_var = name.split('sqrt_')[-1].split(' ')[0].replace('^2','')
            if f'sqrt_{base_var}^2' in name:
                return False  # Élimine sqrt_x^2
        return True

    valid_coefs = coef_df[coef_df['feature'].apply(is_valid_feature)].head(top_coefs)

    return valid_coefs

resultats = analyser_modele_polynomial_final(df_numeric)
display(resultats)
```

```
R²: 0.896
Échantillons utilisés: 80131
```

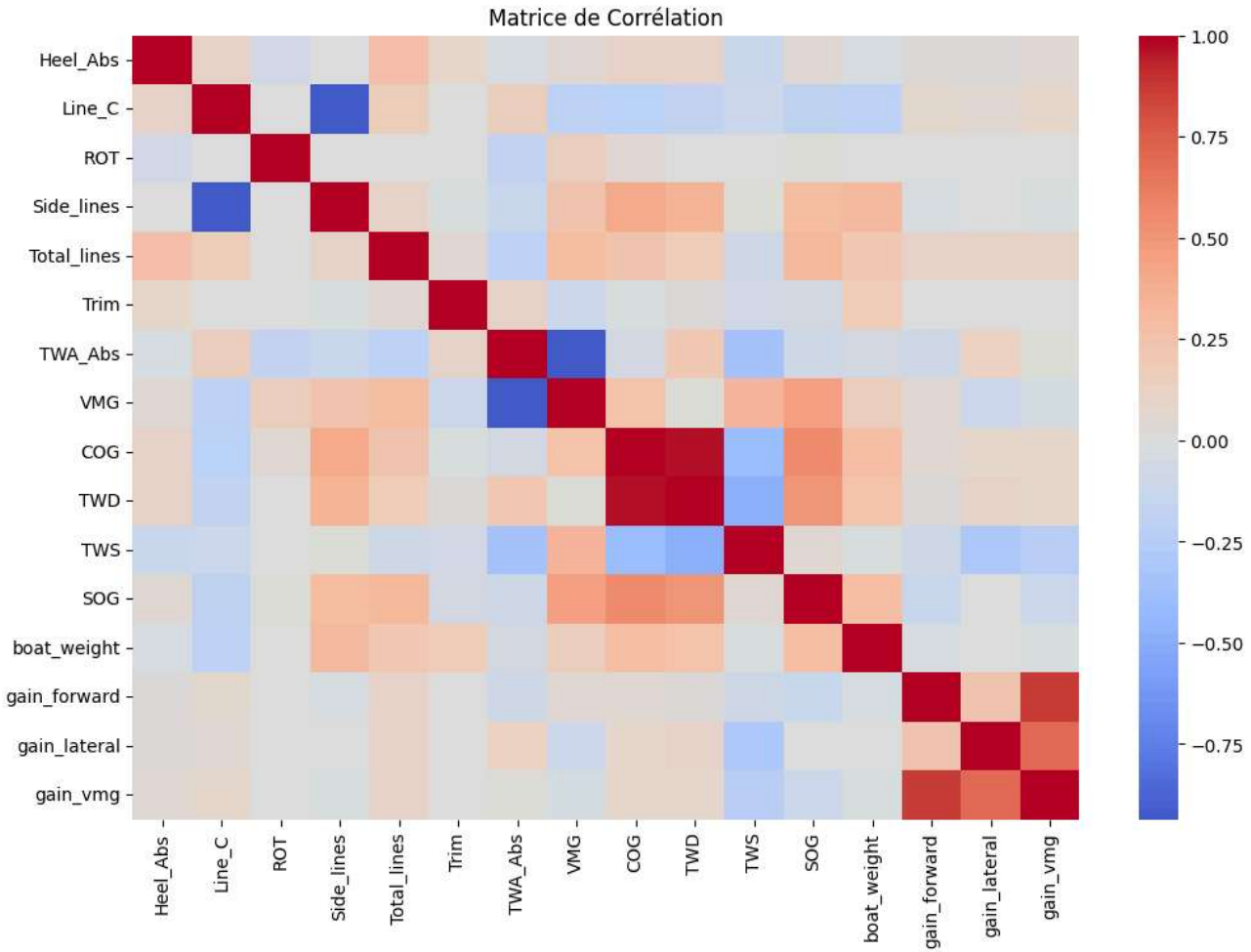
	feature	coefficient
7	TWD	-4.685243e+10
0	Heel_Abs	-5.969228e+09
8	TWS	2.595316e+09
6	COG	1.846250e+09
5	TWA_Abs	1.175460e+09
9	boat_weight	-4.862469e+08
174	boat_weight sqrt_boat_weight	2.365587e+05
164	boat_weight^2	-8.072078e+04
226	sqrt_TWD sqrt_boat_weight	7.540666e+04
151	TWD sqrt_boat_weight	-7.203744e+04
19	sqrt_boat_weight	6.429868e+04
172	boat_weight sqrt_TWD	-4.460727e+04
141	TWD boat_weight	3.893620e+04
17	sqrt_TWD	-3.662378e+04
149	TWD sqrt_TWD	-1.085203e+04
124	TWA_Abs sqrt_boat_weight	-8.670318e+03
223	sqrt_COG sqrt_boat_weight	-8.411285e+03
138	COG sqrt_boat_weight	8.347097e+03
228	sqrt_TWS sqrt_boat_weight	7.805033e+03
93	Side_lines sqrt_boat_weight	7.679783e+03
163	TWS sqrt_boat_weight	-7.667451e+03
219	sqrt_TWA_Abs sqrt_boat_weight	7.053154e+03
208	sqrt_Side_lines sqrt_boat_weight	-6.396072e+03
16	sqrt_COG	6.136401e+03
114	TWA_Abs boat_weight	4.323421e+03
171	boat_weight sqrt_COG	4.188412e+03
173	boat_weight sqrt_TWS	-4.175905e+03
221	sqrt_COG sqrt_TWD	-4.172653e+03
128	COG boat_weight	-4.149939e+03
3	Side_lines	-4.090423e+03

Upwind:

In [6]:
```
# Same analysis, but for downwind and upwind separately
upwind_data = df[df['TWA'] >= 0]
```

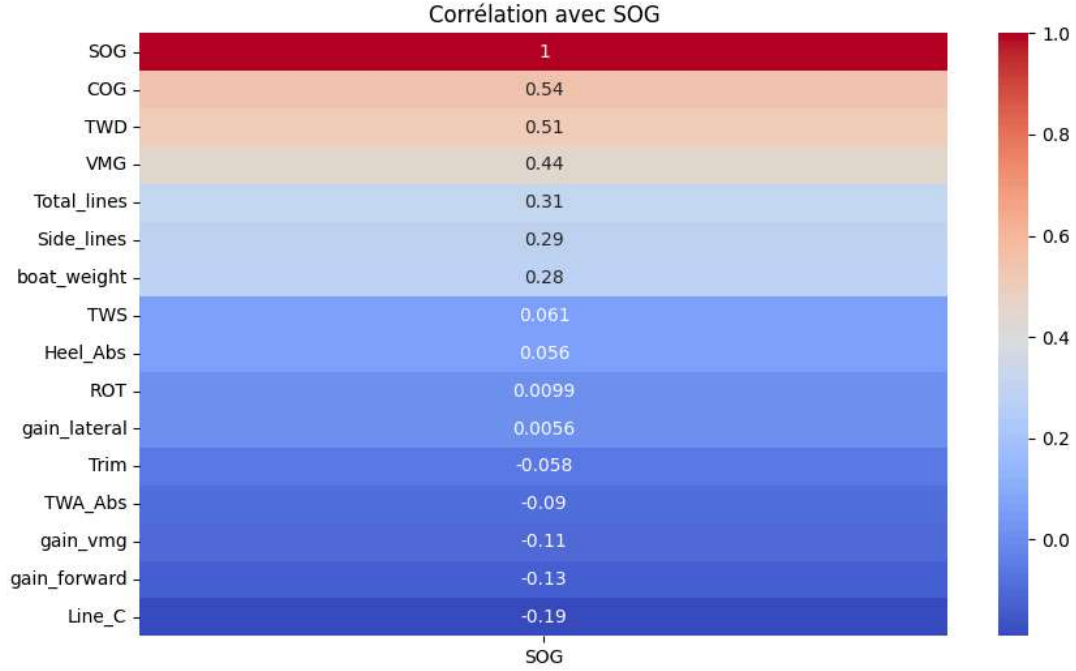
```
# Create a numeric-only version of upwind_data (not df)
df_numeric_upwind = upwind_data.select_dtypes(include=["float64", "int64"]).copy()

"""
drop_cols = [
    "TWA_Abs", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel_Abs", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
"""
drop_cols = [
    "TWA", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
df_numeric_upwind.drop(columns=[c for c in drop_cols if c in df_numeric_upwind.columns], inplace=True)
df_numeric_upwind.dropna(subset=["SOG"], inplace=True)
print(f"Variables utilisées:", df_numeric_upwind.columns.tolist())

afficher_matrice_correlation(df_numeric_upwind)
```

```
Variables utilisées: ['Heel_Abs', 'Line_C', 'ROT', 'Side_lines', 'Total_lines', 'Trim', 'TWA_Abs', 'VMG', 'COG', 'TWD', 'TWS', 'SOG', 'boat_weight', 'gain_forward', 'gain_lateral', 'gain_vmg']
```



In [7]:
```
afficher_correlation_vmg(df_numeric_upwind)
```

```
Corrélation avec SOG :
SOG             1.000000
COG             0.542739
TWD             0.510248
VMG             0.442351
Total_lines     0.313285
Side_lines      0.289328
boat_weight     0.280603
TWS             0.061336
Heel_Abs        0.056305
ROT             0.009906
gain_lateral    0.005635
Trim           -0.058428
TWA_Abs        -0.089706
gain_vmg       -0.105778
gain_forward   -0.133958
Line_C         -0.190149
Name: SOG, dtype: float64
```



In [8]:
```
resultats_anova_upwind = calculer_anova_quadratique(df_numeric_upwind)
resultats_anova_upwind
```

```
Missing values before imputation:
 Heel_Abs           0
Line_C           105
ROT                0
Side_lines      5601
Total_lines     5601
Trim               0
TWA_Abs            0
VMG                0
COG                0
TWD                0
TWS                0
boat_weight        0
gain_forward     113
gain_lateral     113
gain_vmg         113
dtype: int64
Shape after imputation: (48787, 15)
```

Out[8]:

	sum_sq	df	F	PR(>F)
I(TWA_Abs ** 2)	4.931652	1.0	6759.682262	0.000000e+00
I(TWA_Abs ** (1 / 2))	1.022833	1.0	1401.970242	1.234688e-272
I(VMG ** (1 / 2))	0.527121	1.0	722.511092	2.312584e-149
I(TWS ** (1 / 2))	0.120137	1.0	164.668519	3.812791e-37
TWS	0.114292	1.0	156.656915	1.907420e-35
I(TWS ** 2)	0.099268	1.0	136.064217	4.592369e-31
I(COG ** (1 / 2))	0.090299	1.0	123.770872	1.939940e-28
I(COG ** 2)	0.068948	1.0	94.505593	3.729146e-22
I(boat_weight ** (1 / 2))	0.031579	1.0	43.283969	5.187360e-11
boat_weight	0.031123	1.0	42.658929	7.121774e-11
I(boat_weight ** 2)	0.030195	1.0	41.388065	1.357158e-10
I(TWD ** (1 / 2))	0.028415	1.0	38.947928	4.688648e-10
I(gain_vmg ** 2)	0.026933	1.0	36.916327	1.318542e-09
VMG	0.020993	1.0	28.774711	8.473659e-08
I(TWD ** 2)	0.018673	1.0	25.594139	4.354387e-07
gain_lateral	0.017750	1.0	24.329930	8.363207e-07
I(gain_lateral ** (1 / 2))	0.014726	1.0	20.184043	7.182313e-06
I(gain_forward ** (1 / 2))	0.014241	1.0	19.519482	1.015510e-05
I(gain_vmg ** (1 / 2))	0.013812	1.0	18.931671	1.380146e-05
I(Side_lines ** 2)	0.013767	1.0	18.870514	1.424946e-05
I(Heel_Abs ** (1 / 2))	0.013338	1.0	18.282704	1.937538e-05
Heel_Abs	0.012676	1.0	17.374025	3.118552e-05
Side_lines	0.011806	1.0	16.182318	5.832421e-05
I(Heel_Abs ** 2)	0.011462	1.0	15.710902	7.476235e-05
I(gain_forward ** 2)	0.011127	1.0	15.250944	9.529276e-05
I(gain_lateral ** 2)	0.011077	1.0	15.183422	9.875141e-05
I(Side_lines ** (1 / 2))	0.008361	1.0	11.460616	7.160003e-04
gain_forward	0.005391	1.0	7.388614	6.585027e-03
I(Total_lines ** (1 / 2))	0.003785	1.0	5.188610	2.277479e-02
gain_vmg	0.003313	1.0	4.541075	3.313641e-02
I(Trim ** (1 / 2))	0.003296	1.0	4.517441	3.359682e-02
Total_lines	0.003003	1.0	4.116242	4.252290e-02
Trim	0.002844	1.0	3.898124	4.839159e-02
I(Trim ** 2)	0.002634	1.0	3.610986	5.745296e-02

	sum_sq	df	F	PR(>F)
I(VMG ** 2)	0.002362	1.0	3.237004	7.204871e-02
I(Total_lines ** 2)	0.001806	1.0	2.475083	1.157226e-01
ROT	0.001744	1.0	2.390179	1.221590e-01
I(ROT ** (1 / 2))	0.001409	1.0	1.931604	1.646410e-01
I(ROT ** 2)	0.000627	1.0	0.859384	3.539531e-01
I(Line_C ** 2)	0.000579	1.0	0.794006	3.729318e-01
COG	0.000267	1.0	0.365423	5.455357e-01
TWD	0.000260	1.0	0.356333	5.505754e-01
Line_C	0.000201	1.0	0.276120	5.992772e-01
I(Line_C ** (1 / 2))	0.000017	1.0	0.023408	8.784055e-01
TWA_Abs	0.000006	1.0	0.007849	9.294079e-01
Residual	3.890789	5333.0	NaN	NaN

In [9]:
```
resultats_coefs_upwind = analyser_modele_polynomial_final(df_numeric_upwind)
display(resultats_coefs_upwind.head(30))
```

```
R²: 1.000
Échantillons utilisés: 48787
```

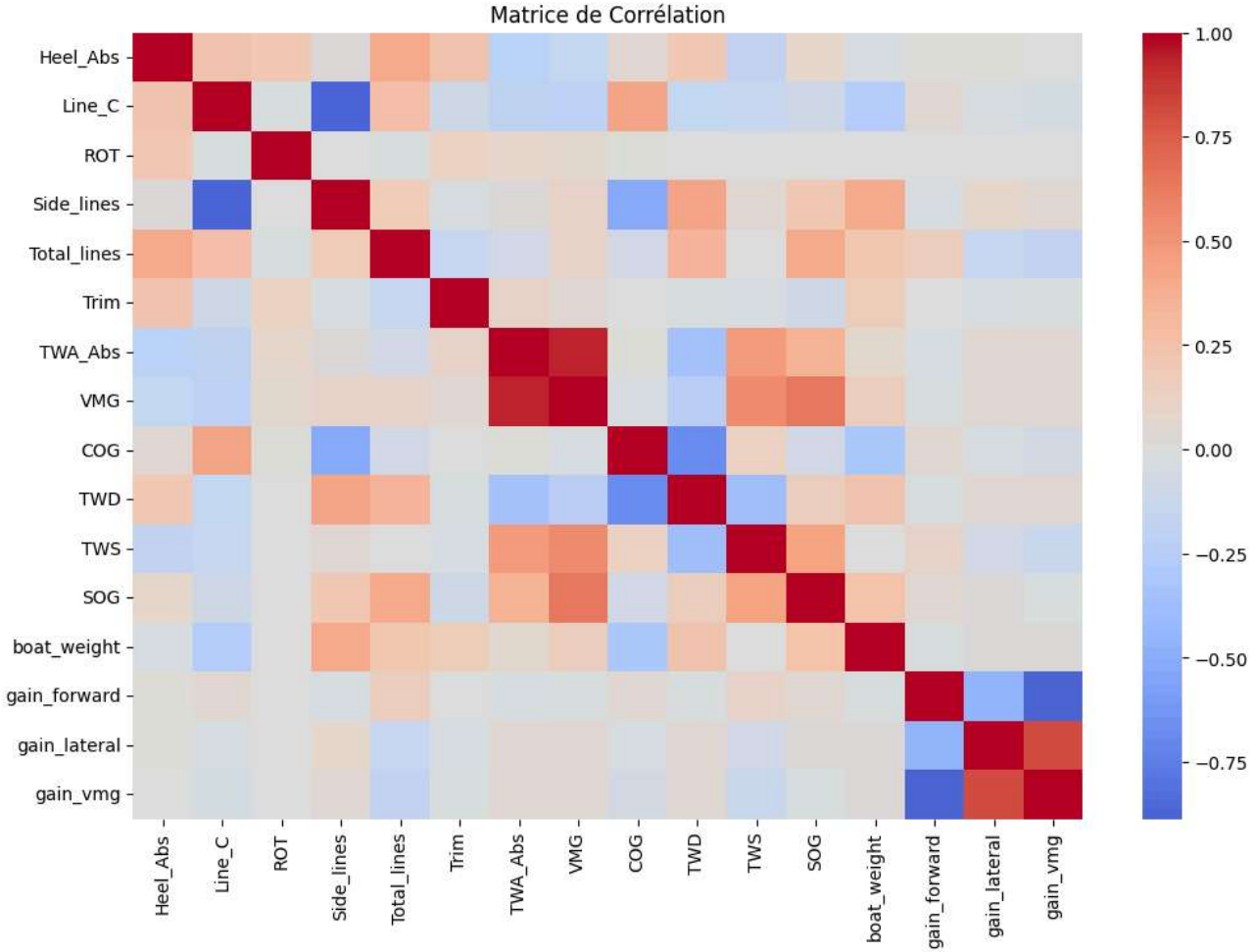
	feature	coefficient
0	Heel_Abs	-4.810955e+08
4	Total_lines	-1.388292e+08
3	Side_lines	4.252558e+07
11	boat_weight	-8.919676e+06
8	COG	4.665623e+06
9	TWD	4.093348e+06
243	COG TWD	2.196443e+06
10	TWS	2.132662e+06
7	VMG	1.398375e+06
264	TWD^2	-1.262035e+06
6	TWA_Abs	1.220931e+06
242	COG^2	-9.526422e+05
198	TWA_Abs TWD	4.754620e+05
197	TWA_Abs COG	-3.736013e+05
195	TWA_Abs^2	-7.878647e+04
258	COG sqrt_TWD	4.222202e+04
278	TWD sqrt_COG	-3.734835e+04
468	sqrt_COG sqrt_TWD	3.069340e+04
276	TWD sqrt_TWA_Abs	-1.656893e+04
255	COG sqrt_TWA_Abs	1.439048e+04
277	TWD sqrt_VMG	1.158388e+04
256	COG sqrt_VMG	-1.154818e+04
453	sqrt_TWA_Abs sqrt_TWD	9.943799e+03
452	sqrt_TWA_Abs sqrt_COG	-9.712432e+03
257	COG sqrt_COG	-8.895251e+03
221	VMG TWD	-7.303249e+03
220	VMG COG	6.927209e+03
320	boat_weight sqrt_boat_weight	3.171522e+03
281	TWD sqrt_boat_weight	-3.148689e+03
210	TWA_Abs sqrt_TWA_Abs	3.105922e+03

Downwind

In [10]:
```
# Same analysis, but for downwind and upwind separately
downwind_data = df[df['TWA'] <= 0]
```

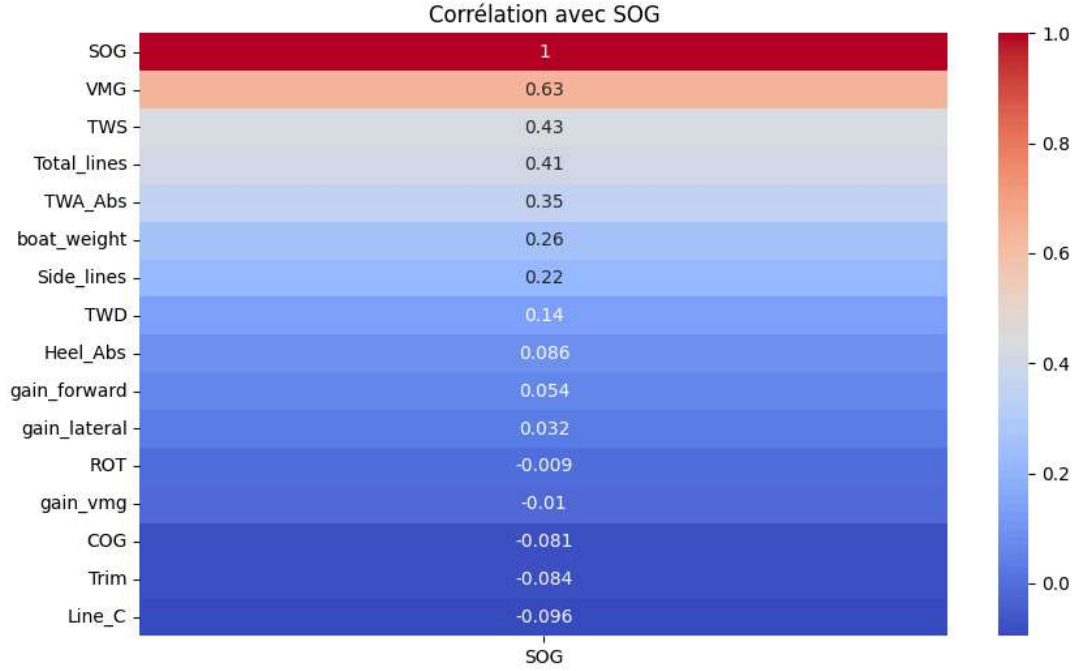
```
# Create a numeric-only version of upwind_data (not df)
df_numeric_downwind = downwind_data.select_dtypes(include=["float64", "int64"]).copy()

"""
drop_cols = [
    "TWA_Abs", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel_Abs", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
"""
drop_cols = [
    "TWA", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
df_numeric_downwind.drop(columns=[c for c in drop_cols if c in df_numeric_downwind.columns], inplace=True)
df_numeric_downwind.dropna(subset=["SOG"], inplace=True)
print(f"Variables utilisées:", df_numeric_downwind.columns.tolist())

afficher_matrice_correlation(df_numeric_downwind)
```

```
Variables utilisées: ['Heel_Abs', 'Line_C', 'ROT', 'Side_lines', 'Total_lines', 'Trim', 'TWA_Abs', 'VMG', 'COG', 'TWD', 'TWS', 'SOG', 'boat_weight', 'gain_forward', 'gain_lateral', 'gain_vmg']
```



In [11]:
```
afficher_correlation_vmg(df_numeric_downwind)
```

```
Corrélation avec SOG :
SOG             1.000000
VMG             0.634291
TWS             0.433601
Total_lines     0.406790
TWA_Abs         0.354902
boat_weight     0.255130
Side_lines      0.215051
TWD             0.140766
Heel_Abs        0.085725
gain_forward    0.054173
gain_lateral    0.032171
ROT            -0.009001
gain_vmg       -0.010182
COG            -0.081012
Trim           -0.083675
Line_C         -0.095684
Name: SOG, dtype: float64
```



In [12]:
```
resultats_anova_downwind = calculer_anova_quadratique(df_numeric_downwind)
resultats_anova_downwind
```

```
Missing values before imputation:
 Heel_Abs           0
Line_C            658
ROT                 0
Side_lines       3844
Total_lines      3844
Trim                0
TWA_Abs             0
VMG                 0
COG                 0
TWD                 0
TWS                 0
boat_weight         0
gain_forward      101
gain_lateral      101
gain_vmg          101
dtype: int64
Shape after imputation: (31344, 15)
```

Out[12]:

	sum_sq	df	F	PR(>F)
I(TWA_Abs ** (1 / 2))	3.901939e-01	1.0	1101.039097	2.915936e-184
TWA_Abs	2.157950e-01	1.0	608.924826	3.086029e-114
I(VMG ** (1 / 2))	1.428213e-01	1.0	403.009409	3.767835e-80
I(TWA_Abs ** 2)	4.658175e-02	1.0	131.443176	2.649277e-29
I(TWD ** 2)	1.216800e-02	1.0	34.335337	5.618639e-09
TWD	1.129915e-02	1.0	31.883638	1.932658e-08
I(TWD ** (1 / 2))	1.099547e-02	1.0	31.026747	2.979359e-08
I(VMG ** 2)	7.321454e-03	1.0	20.659488	5.897500e-06
I(gain_vmg ** 2)	5.216646e-03	1.0	14.720198	1.295377e-04
gain_forward	3.705343e-03	1.0	10.455641	1.247616e-03
gain_vmg	2.960962e-03	1.0	8.355166	3.897668e-03
I(Trim ** (1 / 2))	2.957795e-03	1.0	8.346229	3.916785e-03
I(COG ** 2)	2.510062e-03	1.0	7.082828	7.860281e-03
Trim	2.234834e-03	1.0	6.306197	1.212929e-02
COG	1.804962e-03	1.0	5.093196	2.415398e-02
I(gain_forward ** 2)	1.754178e-03	1.0	4.949893	2.623094e-02
VMG	1.547642e-03	1.0	4.367095	3.679658e-02
I(Trim ** 2)	1.493451e-03	1.0	4.214181	4.024932e-02
I(COG ** (1 / 2))	9.498992e-04	1.0	2.680401	1.017863e-01
I(TWS ** (1 / 2))	8.776727e-04	1.0	2.476594	1.157495e-01
TWS	8.595804e-04	1.0	2.425542	1.195692e-01
I(TWS ** 2)	6.936479e-04	1.0	1.957318	1.619938e-01
I(Side_lines ** 2)	4.927831e-04	1.0	1.390523	2.384917e-01
ROT	4.852758e-04	1.0	1.369339	2.420993e-01
I(Line_C ** 2)	4.544134e-04	1.0	1.282252	2.576500e-01
Side_lines	3.929784e-04	1.0	1.108896	2.924806e-01
gain_lateral	3.747045e-04	1.0	1.057332	3.039796e-01
I(ROT ** (1 / 2))	3.480652e-04	1.0	0.982161	3.218153e-01
I(ROT ** 2)	2.693174e-04	1.0	0.759953	3.834737e-01
Line_C	2.673988e-04	1.0	0.754539	3.851729e-01
I(Side_lines ** (1 / 2))	1.015320e-04	1.0	0.286500	5.925462e-01
I(gain_lateral ** 2)	9.812081e-05	1.0	0.276875	5.988297e-01
I(boat_weight ** (1 / 2))	8.217441e-05	1.0	0.231878	6.302008e-01
I(gain_lateral ** (1 / 2))	7.066173e-05	1.0	0.199391	6.552728e-01

	sum_sq	df	F	PR(>F)
I(gain_forward ** (1 / 2))	6.927178e-05	1.0	0.195469	6.584626e-01
boat_weight	6.525812e-05	1.0	0.184144	6.678938e-01
I(gain_vmg ** (1 / 2))	5.051756e-05	1.0	0.142549	7.058098e-01
I(Total_lines ** (1 / 2))	4.658307e-05	1.0	0.131447	7.169843e-01
I(Heel_Abs ** 2)	3.732500e-05	1.0	0.105323	7.455756e-01
I(boat_weight ** 2)	3.661233e-05	1.0	0.103312	7.479329e-01
Heel_Abs	3.402245e-05	1.0	0.096004	7.567194e-01
Total_lines	3.336756e-05	1.0	0.094156	7.589989e-01
I(Heel_Abs ** (1 / 2))	3.074150e-05	1.0	0.086746	7.683941e-01
I(Total_lines ** 2)	1.571194e-05	1.0	0.044336	8.332573e-01
I(Line_C ** (1 / 2))	1.308320e-07	1.0	0.000369	9.846728e-01
Residual	5.684367e-01	1604.0	NaN	NaN

In [13]:
```
resultats_coefs_downwind = analyser_modele_polynomial_final(df_numeric_downwind)
display(resultats_coefs_downwind.head(30))
```

```
R²: 1.000
Échantillons utilisés: 31344
```

	feature	coefficient
0	Heel_Abs	-3.113761e+09
4	Total_lines	-9.812618e+08
3	Side_lines	3.939803e+08
7	VMG	1.122440e+08
11	boat_weight	7.239214e+07
8	COG	6.592320e+07
9	TWD	4.138481e+07
6	TWA_Abs	3.711885e+07
10	TWS	1.406130e+07
210	TWA_Abs sqrt_TWA_Abs	-4.105529e+03
320	boat_weight sqrt_boat_weight	-2.005278e+03
21	sqrt_TWA_Abs	-1.377769e+03
195	TWA_Abs^2	1.253261e+03
305	boat_weight^2	6.808837e+02
26	sqrt_boat_weight	-5.645252e+02
451	sqrt_TWA_Abs sqrt_VMG	-5.148888e+02
455	sqrt_TWA_Abs sqrt_boat_weight	-3.602296e+02
215	TWA_Abs sqrt_boat_weight	3.318732e+02
211	TWA_Abs sqrt_VMG	3.043014e+02
315	boat_weight sqrt_TWA_Abs	2.411735e+02
22	sqrt_VMG	2.231965e+02
468	sqrt_COG sqrt_TWD	-1.978905e+02
200	TWA_Abs boat_weight	-1.873357e+02
233	VMG sqrt_TWA_Abs	1.864948e+02
258	COG sqrt_TWD	1.694029e+02
255	COG sqrt_TWA_Abs	1.287707e+02
23	sqrt_COG	1.206296e+02
278	TWD sqrt_COG	1.180148e+02
452	sqrt_TWA_Abs sqrt_COG	-1.152378e+02
196	TWA_Abs VMG	-1.042608e+02

In [14]:
```
import scipy.stats as stats

# Perform a t-test between upwind and downwind data for the "SOG" variable
def t_test_between_upwind_downwind(df_upwind, df_downwind, target="SOG"):
    # Perform t-test for the target variable "SOG" between the two datasets
    t_stat, p_value = stats.ttest_ind(df_upwind[target].dropna(), df_downwind[target].dropna())
```

```
    print(f"T-statistic: {t_stat:.3f}, p-value: {p_value:.3f}")

    # If p-value is less than 0.05, the difference is statistically significant
    if p_value < 0.05:
        print("The difference is statistically significant, keeping data split.")
        return True  # Keep data split
    else:
        print("The difference is not statistically significant, keeping data combined.")
        return False  # Keep data combined

# Call the t-test function
split_data = t_test_between_upwind_downwind(df_numeric_upwind, df_numeric_downwind)

# Based on the result of the t-test, split or combine data
if split_data:
    # If the t-test shows a significant difference, keep data split
    # The rest of your code for analysis on `df_numeric_upwind` and `df_numeric_downwind` continues
    print("Proceeding with separate analyses for upwind and downwind data.")
else:
    # If the t-test does not show a significant difference, combine the data
    df_numeric_combined = pd.concat([df_numeric_upwind, df_numeric_downwind], axis=0)
    print("Proceeding with combined analysis.")
    # You can call your existing functions on the combined data
    resultats_anova_combined = calculer_anova_quadratique(df_numeric_combined)
    resultats_coefs_combined = analyser_modele_polynomial_final(df_numeric_combined)
    display(resultats_anova_combined)
    display(resultats_coefs_combined)
```

```
T-statistic: -482.303, p-value: 0.000
The difference is statistically significant, keeping data split.
Proceeding with separate analyses for upwind and downwind data.
```

In [ ]: