DEEP STRUCTURAL MINDS SciML PROJECT

Texas has the largest number of beam-slab bridges in the country according to National Bridge Inventory (FHWA 2022). Railings or crash barriers are provided on either side of deck for crash protection and to contain the colliding vehicle within the bridge. In recent times, there is a necessity to provide heavier and taller railings or sound walls for higher crash rating or to reduce noise in bridges crossing urban residential areas. These heavy railings and sound walls weight two to three times that of normal railings. The Texas Department of Transportaion (TxDOT) recommends to distribute railing dead loads to two or three girders for normal railings (TxDOT 2022). But this recommendation is not applicable in case of heavy railings and sound walls. So designers need to perform finite element analysis in such cases.

The project team has performed parametric Finite Element (FE) analyses for various geometries of Pre-Stressed Concrete (PSC) I-girder straight bridges. The bending moments and bearing reactions are of primary importance in structural design of these bridges. There are 8 standard PSC I-girders used in Texas. The span-to-depth ratios typically range between 15 to 30. All these bridges have 4 girder lines with girder center-to-center spacing varying from 6 to 12 ft. The minimum width of overhang varies from half of girder top flange width (zero overhang from edge of girder) to a maximum of 3.5 ft. Deck slab was assumed to be of uniform thickness throughout the entire width of the bridge.Deck slab thickness values of 8.5 in. and 12 in. were used in the models. The concrete grade for the deck and girders were assumed to be 4 ksi. and 7 ksi., respectively. A railing load of 1000 lb/ft. was applied on the left edge of the deck distributed over the width of the railing along the full length of the bridge.

The bending moment and bearing reactions are dependent on the stiffness properties of the bridge. Hence, they are dependent on the material and geometric properties of the bridge. In this project, we intend to develop a Machine Learning (ML) model that can predict the design forces for any given PSC I-girder bridge geometry. The raw data is imported from github using pandas.

ML models work well on normalized data, therefore the bending moments and reactions were normalized before training. Line analysis was performed for each girder with the entire weight of the railing assumed to be applied along the centerline of the girder. Since, only simply-supported bridges were analyzed the results for line analysis could be computed easily by closed-form solutions. The ratio of force obtained from FE analysis to the line analysis is defined as distribution factor. These factors indicated the proportion of load transferred to each girder, thereby the designer can optimize the design by choosing appropriate geometry. The distribution factors calculated from bending moments is called as Moment Distribution Factor (df_M) and that from the bearing reactions is called as Shear Distribution Factor (df_V). These distribution factors are the output variables for the ML model. Hence, there are 8 outputs for each 4-girder bridge system. The variation of these distribution factors for G1 (left exterior) and G2 (first left-interior) girders are plotted below. These girders were of importance because they are close to the railing load.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm.notebook import tqdm

df = pd.read_csv('https://raw.githubusercontent.com/bhushanrajs/sciml_project/main/analys
df = df.dropna()
print(df)
```

```
                               Model Name Girder Type  Nb     L    S  w_oh  ts_U  \
0       Tx28-L_45-Nb_4-S_60-O_15-ts_85        Tx28   4   540   72    18   8.5
1      Tx28-L_45-Nb_4-S_60-O_15-ts_100        Tx28   4   540   72    18  10.0
2       Tx28-L_45-Nb_4-S_60-O_20-ts_85        Tx28   4   540   72    24   8.5
3      Tx28-L_45-Nb_4-S_60-O_20-ts_100        Tx28   4   540   72    24  10.0
4       Tx28-L_45-Nb_4-S_60-O_25-ts_85        Tx28   4   540   72    30   8.5
...                                ...         ...  ..   ...  ...   ...   ...
2060   Tx84-L_160-Nb_4-S_80-O_35-ts_85        Tx84   4  1920   96    42   8.5
2061   Tx84-L_160-Nb_4-S_90-O_30-ts_85        Tx84   4  1920  108    36   8.5
2062   Tx84-L_160-Nb_4-S_90-O_35-ts_85        Tx84   4  1920  108    42   8.5
2063  Tx84-L_160-Nb_4-S_100-O_25-ts_85        Tx84   4  1920  120    30   8.5
2064  Tx84-L_160-Nb_4-S_100-O_35-ts_85        Tx84   4  1920  120    42   8.5

      ts_O  fc_deck  fc_girder  ...       G3-A2-Y       G4-A2-Y  G1-A1-Z  \
0      8.5     4000       7000  ...    880.440674  -1000.869019        0
1     10.0     4000       7000  ...   1138.782349  -1364.887085        0
2      8.5     4000       7000  ...    829.689697  -1071.558472        0
3     10.0     4000       7000  ...   1079.834351  -1457.175049        0
4      8.5     4000       7000  ...    757.181519  -1138.819336        0
...    ...      ...        ...  ...           ...           ...      ...
2060   8.5     4000       7000  ...  10792.476560 -12379.569340        0
2061   8.5     4000       7000  ...   8513.214844 -10040.167970        0
2062   8.5     4000       7000  ...   8378.083984 -10616.997070        0
2063   8.5     4000       7000  ...   6496.878418  -8156.199707        0
2064   8.5     4000       7000  ...   6189.978027  -9032.711914        0

      G2-A1-Z  G3-A1-Z  G4-A1-Z  G1-A2-Z  G2-A2-Z  G3-A2-Z  G4-A2-Z
0           0        0        0        0        0        0        0
1           0        0        0        0        0        0        0
2           0        0        0        0        0        0        0
3           0        0        0        0        0        0        0
4           0        0        0        0        0        0        0
...       ...      ...      ...      ...      ...      ...      ...
2060        0        0        0        0        0        0        0
2061        0        0        0        0        0        0        0
```

```
2062      0      0      0      0      0      0      0
2063      0      0      0      0      0      0      0
2064      0      0      0      0      0      0      0

[2065 rows x 66 columns]
```

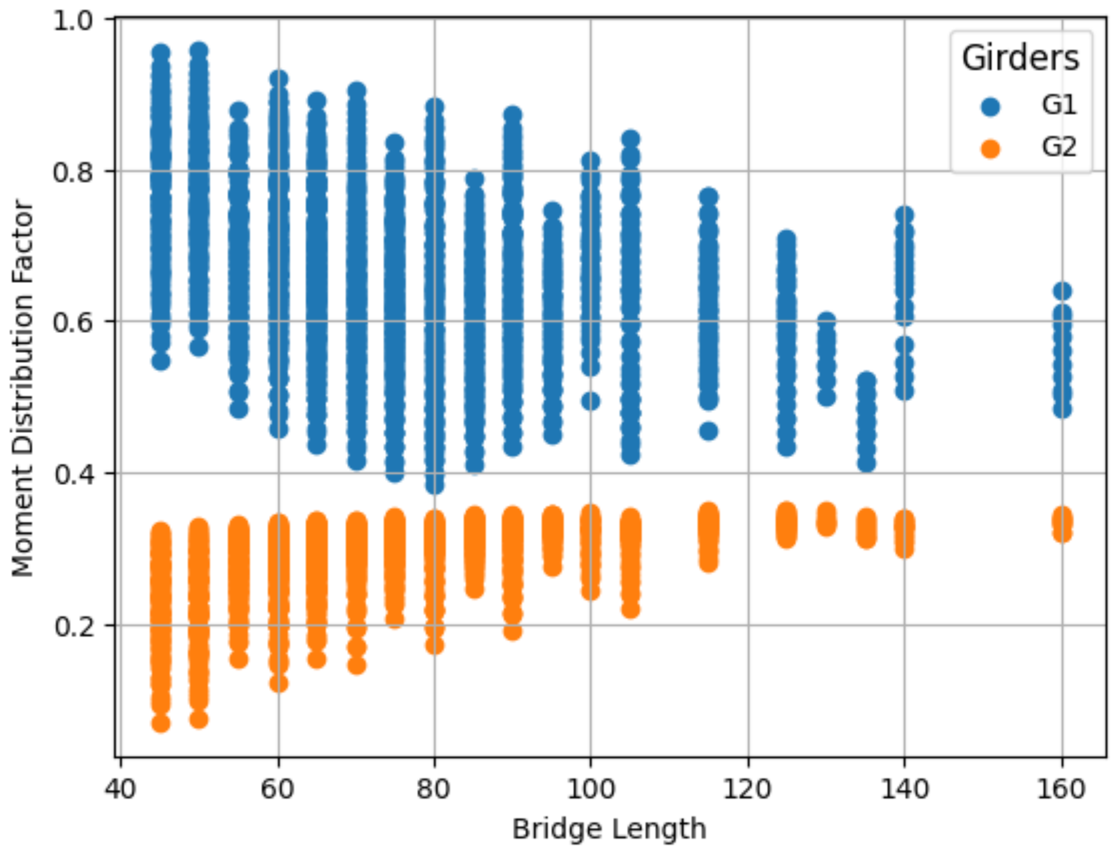
```
tx_girders = {'Tx28' : {'D' : 28.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx34' : {'D' : 34.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx40' : {'D' : 40.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx46' : {'D' : 46.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx54' : {'D' : 54.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx62' : {'D' : 62.0, 'b1' : 42.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx70' : {'D' : 70.0, 'b1' : 42.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx84' : {'D' : 84.0, 'b1' : 58.0, 'b2' : 8.0, 'b3' : 38.0, 'b4' : 3.0, 'b5
              }

# bridge geometry data
L = df['L']/12 # span length in ft.
S = df['S']/12 # girder spacing in ft.
w_oh = df['w_oh']/12 # overhang width in ft.
ts = df['ts_U']/12 # thickness of overhang in ft.
girder = df['Girder Type']
girder_depth = []
for _, girder_type in girder.items():
  girder_depth.append(tx_girders[girder_type]['D'])
df['D'] = girder_depth

# intensity of railing dead load in kip/ft
b_rail = df['b_rail_left'] # width of railing
q_rail = (df['q_rail_left'] * b_rail * 12)/1000

# max bending moment in exterior girder G1 & interior girder G2 in kip-ft
bm1 = df['G1 - max_bm']/(1000*12)
bm2 = df['G2 - max_bm']/(1000*12)
bm3 = df['G3 - max_bm']/(1000*12)
bm4 = df['G4 - max_bm']/(1000*12)

# reaction in girders G1 & G2 in kip. (only A1 taken due to symmetry)
r1 = df['G1-A1-Y']/1000
r2 = df['G2-A1-Y']/1000
r3 = df['G3-A1-Y']/1000
r4 = df['G4-A1-Y']/1000

# line analysis with full railing load assumed to be applied on a girder
bm_line = q_rail * (L - 2*9/12)**2 / 8
r_line = q_rail * L / 2


# normalizing bending moments with respect to line analysis
n_bm1 = bm1 / bm_line
```

```
n_bm2 = bm2 / bm_line
n_bm3 = bm3 / bm_line
n_bm4 = bm4 / bm_line

# normalizing vertical reactions with respect to line analysis
n_r1 = r1 / r_line
n_r2 = r2 / r_line
n_r3 = r3 / r_line
n_r4 = r4 / r_line


df['n_bm1'] = n_bm1
df['n_bm2'] = n_bm2
df['n_bm3'] = n_bm3
df['n_bm4'] = n_bm4
df['n_r1'] = n_r1
df['n_r2'] = n_r2
df['n_r3'] = n_r3
df['n_r4'] = n_r4

print(n_bm1 + n_bm2 + n_bm3 + n_bm4)
print(n_r1 + n_r2 + n_r3 + n_r4)
```

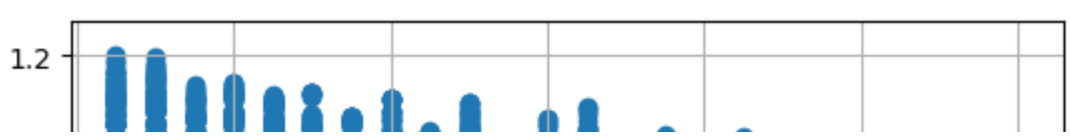
```
0       0.990695
1       0.986348
2       0.998870
3       0.995075
4       1.007073
          ...
2060    1.039071
2061    1.045995
2062    1.054395
2063    1.048749
2064    1.063360
Length: 2065, dtype: float64
0       1.000001
1       1.000001
2       1.000001
3       1.000001
4       1.000001
          ...
2060    1.000001
2061    1.000001
2062    1.000001
2063    1.000001
2064    1.000001
Length: 2065, dtype: float64
```

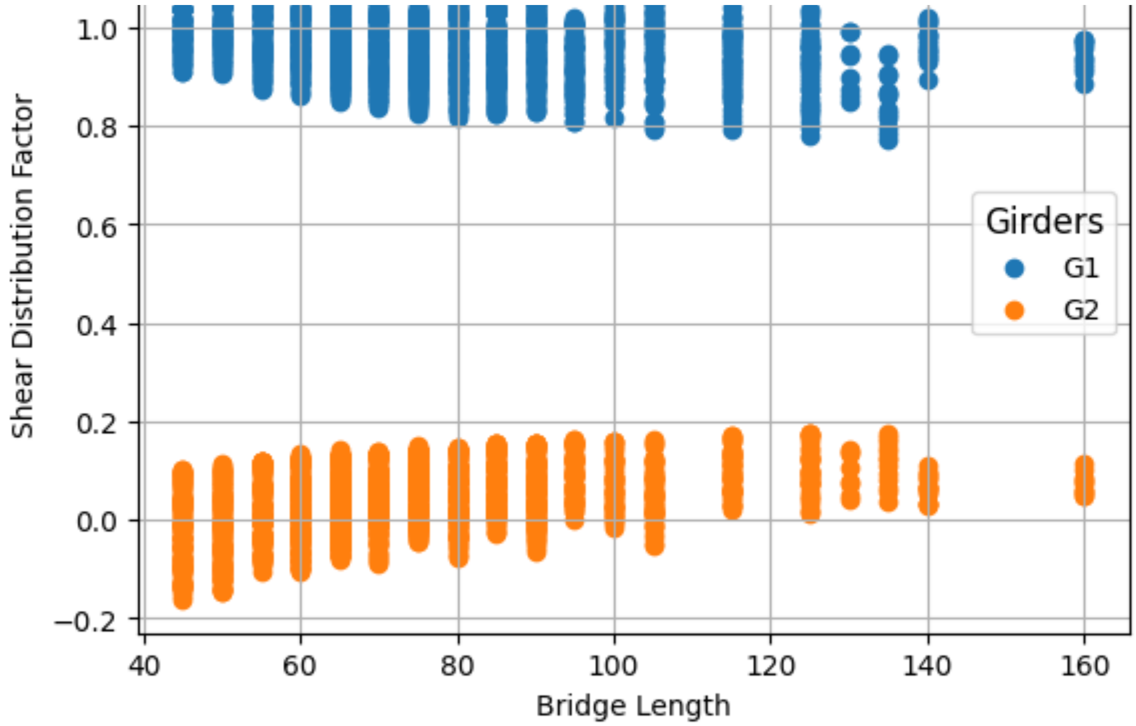
```
# Plot moment distribution factors
plt.scatter(L,n_bm1, label = "G1")
plt.scatter(L,n_bm2, label = "G2")
```

```
# axis labels
plt.xlabel(r'Bridge Length')
plt.ylabel(r'Moment Distribution Factor')

# display a legend, set its title
leg = plt.legend()
leg.set_title('Girders', prop={'size':12})

plt.grid() # show a grid
```




```
# Plot shear distribution factors
plt.scatter(L,n_r1, label = "G1")
plt.scatter(L,n_r2, label = "G2")

# axis labels
plt.xlabel(r'Bridge Length')
plt.ylabel(r'Shear Distribution Factor')

# display a legend, set its title
leg = plt.legend()
leg.set_title('Girders', prop={'size':12})

plt.grid() # show a grid
```





Using a multi-layer perceptron with five features and three hidden layers containing 32 neurons each to predict the distribution factors for bending moments and reaction forces for the exterior and interior girders (4-girder line)

```
df_new = df[['L', 'S', 'w_oh', 'ts_U','D', 'n_bm1','n_bm2', 'n_bm3','n_bm4','n_r1', 'n_r2
df_new.head()
```

	L	S	w_oh	ts_U	D	n_bm1	n_bm2	n_bm3	n_bm4	n_r1	n_r2
0	540	72	18	8.5	28.0	0.570854	0.319484	0.099727	0.000631	0.908724	0.096629
1	540	72	18	10.0	28.0	0.549410	0.323670	0.112149	0.001120	0.908414	0.101636
2	540	72	24	8.5	28.0	0.602907	0.306543	0.088845	0.000576	0.973940	0.036811
3	540	72	24	10.0	28.0	0.579155	0.313099	0.101768	0.001053	0.973641	0.043130
4	540	72	30	8.5	28.0	0.634685	0.293845	0.078021	0.000522	1.038466	-0.021503

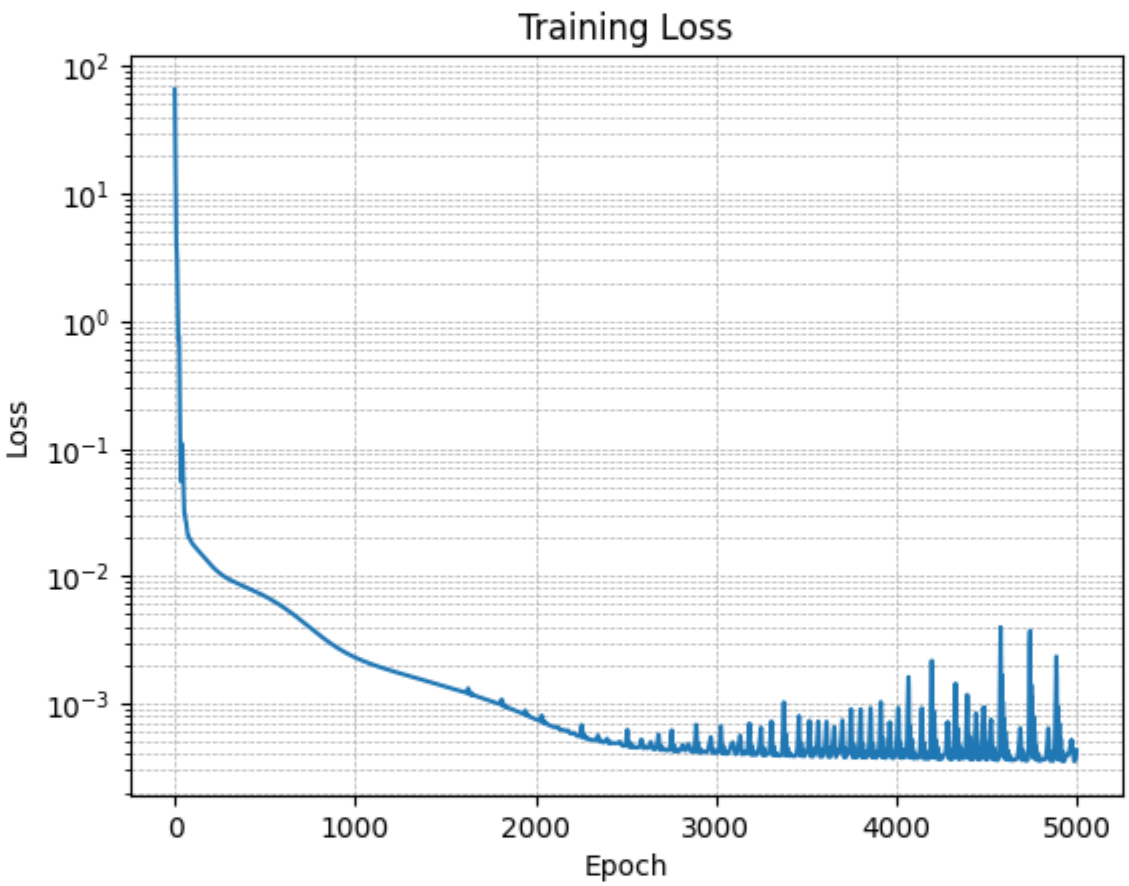
```
X = df_new.copy(deep=True)
y = df_new[['n_bm1','n_bm2', 'n_bm3','n_bm4','n_r1', 'n_r2','n_r3', 'n_r4']]

tensor_y = torch.tensor(y.values, dtype=torch.float32)

X_train_target, X_val_test_target, y_train, y_val_test = train_test_split(X, y, test_size
```

```
X_test_target, X_val_target, y_test, y_val = train_test_split(X_val_test_target, y_val_te

X_train = X_train_target.drop(['n_bm1','n_bm2', 'n_bm3','n_bm4','n_r1', 'n_r2','n_r3', 'n
X_test = X_test_target.drop(['n_bm1','n_bm2', 'n_bm3','n_bm4','n_r1', 'n_r2','n_r3', 'n_r
X_val = X_val_target.drop(['n_bm1','n_bm2', 'n_bm3','n_bm4','n_r1', 'n_r2','n_r3', 'n_r4'

tensor_y_train = torch.tensor(y_train.values, dtype=torch.float32)
tensor_y_val = torch.tensor(y_val.values, dtype=torch.float32)
tensor_y_test = torch.tensor(y_test.values, dtype=torch.float32)

tensor_X_train = torch.tensor(X_train.values, dtype=torch.float32)
tensor_X_test = torch.tensor(X_test.values, dtype=torch.float32)
tensor_X_val = torch.tensor(X_val.values, dtype=torch.float32)


model = nn.Sequential(
    nn.Linear(5, 32),  # Input layer with 5 features and 32 units
    nn.ReLU(),        # Activation function (you can choose other activation functions)
    nn.Linear(32, 32),
    nn.ReLU(),
    nn.Linear(32, 32),
    nn.ReLU(),
    nn.Linear(32, 32),
    nn.ReLU(),
    nn.Linear(32, 8)   # Output layer with 64 units and 2 output units
)

# Training
optimizer = optim.Adam(model.parameters())
losses = []
epochs = 5000
for epoch in tqdm(range(epochs), desc='Model training progress'):
    y_pred = model(tensor_X_train)

    # Define the loss function for each output
    loss_fn_output = nn.MSELoss()
    total_loss = loss_fn_output(y_pred, tensor_y_train)

    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()
    losses.append(total_loss.item())
```

Model training progress: 5000/5000 [00:21<00:00,

```
# Plot the loss on a semilog scale
plt.figure()
plt.semilogy(losses)
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.title('Training Loss')
plt.grid(True, which="both", ls="--", linewidth=0.5)
plt.show()

print(f"Training loss is: {total_loss}")
```




```
Training loss is: 0.00043226347770541906
```

```
y_pred_test = model(tensor_X_test)
loss_test = loss_fn_output(y_pred_test, tensor_y_test)

print(f"Test loss is: {loss_test}")
```

```
Test loss is: 0.00046123474021442235
```

Symbolic Regression

```
%%shell
set -e

#---------------------------------------------------#
JULIA_VERSION="1.8.5"
export JULIA_PKG_PRECOMPILE_AUTO=0
#---------------------------------------------------#

if [ -z `which julia` ]; then
  # Install Julia
  JULIA_VER=`cut -d '.' -f -2 <<< "$JULIA_VERSION"`
  echo "Installing Julia $JULIA_VERSION on the current Colab Runtime..."
  BASE_URL="https://julialang-s3.julialang.org/bin/linux/x64"
  URL="$BASE_URL/$JULIA_VER/julia-$JULIA_VERSION-linux-x86_64.tar.gz"
  wget -nv $URL -O /tmp/julia.tar.gz # -nv means "not verbose"
  tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
  rm /tmp/julia.tar.gz

  echo "Installing PyCall.jl..."
  julia -e 'using Pkg; Pkg.add("PyCall"); Pkg.build("PyCall")'
  julia -e 'println("Success")'

fi
```

```
Installing Julia 1.8.5 on the current Colab Runtime...
2023-12-04 05:13:45 URL:https://julialang-s3.julialang.org/bin/linux/x64/1.8/julia-1.
Installing PyCall.jl...
  Installing known registries into `~/.julia`
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
   Installed VersionParsing ── v1.3.0
   Installed MacroTools ────── v0.5.11
   Installed Parsers ───────── v2.8.0
   Installed Conda ─────────── v1.10.0
   Installed PyCall ────────── v1.96.2
   Installed Preferences ───── v1.4.1
   Installed PrecompileTools ─ v1.2.0
   Installed JSON ──────────── v0.21.4
    Updating `~/.julia/environments/v1.8/Project.toml`
  [438e738f] + PyCall v1.96.2
    Updating `~/.julia/environments/v1.8/Manifest.toml`
  [8f4d0f93] + Conda v1.10.0
  [682c06a0] + JSON v0.21.4
  [1914dd2f] + MacroTools v0.5.11
  [69de0a69] + Parsers v2.8.0
  [aea7be01] + PrecompileTools v1.2.0
  [21216c6a] + Preferences v1.4.1
```

```
  [438e738f] + PyCall v1.96.2
  [81def892] + VersionParsing v1.3.0
  [0dad84c5] + ArgTools v1.1.1
  [56f22d72] + Artifacts
  [2a0f44e3] + Base64
  [ade2ca70] + Dates
  [f43a241f] + Downloads v1.6.0
  [7b1f6079] + FileWatching
  [b27032c2] + LibCURL v0.6.3
  [8f399da3] + Libdl
  [37e2e46d] + LinearAlgebra
  [d6f4376e] + Markdown
  [a63ad114] + Mmap
  [ca575930] + NetworkOptions v1.2.0
  [de0858da] + Printf
  [9a3f8284] + Random
  [ea8e919c] + SHA v0.7.0
  [9e88b42a] + Serialization
  [fa267f1f] + TOML v1.0.0
  [cf7118a7] + UUIDs
  [4ec0a83e] + Unicode
  [e66e0078] + CompilerSupportLibraries_jll v1.0.1+0
  [deac9b47] + LibCURL_jll v7.84.0+0
  [29816b5a] + LibSSH2_jll v1.10.2+0
  [c8ffd9c3] + MbedTLS_jll v2.28.0+0
  [14a3606d] + MozillaCACerts_jll v2022.2.1
  [4536629a] + OpenBLAS_jll v0.3.20+0
  [83775a58] + Zlib_jll v1.2.12+3
  [8e850b90] + libblastrampoline_jll v5.1.1+0
  [8e850ede] + nghttp2_jll v1.48.0+0
    Building Conda ─→ `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/51
    Building PyCall → `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/1c
    Building Conda ─→ `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/51
    Building PyCall → `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/1c
Success
```

Install PySR and PyTorch-Lightning:

```
%pip install -Uq pysr pytorch_lightning --quiet
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 1.2 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 776.9/776.9 kB 8.4 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 68.7/68.7 kB 10.2 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 806.1/806.1 kB 45.6 MB/s eta 0:00:00
```

```
from julia import Julia

julia = Julia(compiled_modules=False, threads="auto")
from julia import Main
from julia.tools import redirect_output_streams

redirect_output_streams()
```

```
import pysr

# We don't precompile in colab because compiled modules are incompatible static Python li
pysr.install(precompile=False)
```

```
Julia Version 1.8.5
Commit 17cfb8e65ea (2023-01-08 06:45 UTC)
Platform Info:
  OS: Linux (x86_64-linux-gnu)
      Ubuntu 22.04.3 LTS
  uname: Linux 5.15.120+ #1 SMP Wed Aug 30 11:19:59 UTC 2023 x86_64 x86_64
  CPU: Intel(R) Xeon(R) CPU @ 2.30GHz:
              speed         user         nice          sys         idle          irq
       #1  2299 MHz        732 s          0 s        137 s        820 s          0 s
       #2  2299 MHz        878 s          0 s        155 s        658 s          0 s
  Memory: 12.6783447265625 GB (11510.1015625 MB free)
  Uptime: 176.74 sec
  Load Avg:  2.07  0.98  0.39
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-13.0.1 (ORCJIT, haswell)
  Threads: 1 on 2 virtual cores
Environment:
  LD_LIBRARY_PATH = /usr/lib64-nvidia
  JULIA_PROJECT = @pysr-0.16.3
  JULIA_PKG_PRECOMPILE_AUTO = 0
  TCLLIBPATH = /usr/share/tcltk/tcllib1.20
  HOME = /root
  PYTHONPATH = /env/python
  LIBRARY_PATH = /usr/local/cuda/lib64/stubs
  PATH = /opt/bin:/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/loca
  COLAB_DEBUG_ADAPTER_MUX_PATH = /usr/local/bin/dap_multiplexer
  TERM = xterm-color
[ Info: Julia version info
[ Info: Julia executable: /usr/local/bin/julia
[ Info: Trying to import PyCall...
┌ Info: PyCall is already installed and compatible with Python executable.
│
│ PyCall:
│     python: /usr/bin/python3
│     libpython: /usr/lib/x86_64-linux-gnu/libpython3.10.so.1.0
│ Python:
│     python: /usr/bin/python3
└     libpython:
    Updating registry at `~/.julia/registries/General.toml`
     Cloning git-repo `https://github.com/MilesCranmer/SymbolicRegression.jl`
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
   Installed Tricks ─────────────────────────── v0.1.8
   Installed ScientificTypesBase ────────────── v3.0.0
   Installed BitTwiddlingConvenienceFunctions ─ v0.1.5
   Installed SIMDTypes ──────────────────────── v0.1.0
   Installed DiffRules ──────────────────────── v1.15.1
   Installed DynamicExpressions ─────────────── v0.13.1
   Installed LayoutPointers ─────────────────── v0.1.15
```

```
    Installed CpuId ──────────────────────────── v0.3.1
    Installed MLJModelInterface ──────────────── v1.9.3
    Installed DiffResults ────────────────────── v1.1.0
    Installed VectorizationBase ──────────────── v0.21.65
    Installed StatisticalTraits ──────────────── v3.2.0
    Installed PositiveFactorizations ─────────── v0.2.4
    Installed CPUSummary ─────────────────────── v0.2.4
    Installed PDMats ─────────────────────────── v0.11.30
```

```
import sympy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from pysr import PySRRegressor
from sklearn.model_selection import train_test_split

df = pd.read_csv('https://raw.githubusercontent.com/bhushanrajs/sciml_project/main/analys

tx_girders = {'Tx28' : {'D' : 28.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx34' : {'D' : 34.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx40' : {'D' : 40.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx46' : {'D' : 46.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx54' : {'D' : 54.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx62' : {'D' : 62.0, 'b1' : 42.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx70' : {'D' : 70.0, 'b1' : 42.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx84' : {'D' : 84.0, 'b1' : 58.0, 'b2' : 8.0, 'b3' : 38.0, 'b4' : 3.0, 'b5
              }

# bridge geometry data
L = df['L']/12 # span length in ft.
S = df['S']/12 # girder spacing in ft.
w_oh = df['w_oh']/12 # overhang width in ft.
ts = df['ts_U']/12 # thickness of overhang in ft.
girder = df['Girder Type']
D = []
for _, girder_type in girder.items():
  D.append(tx_girders[girder_type]['D'])
df['D'] = D

# intensity of railing dead load in kip/ft
b_rail = df['b_rail_left'] # width of railing
q_rail = (df['q_rail_left'] * b_rail * 12)/1000

# max bending moment in exterior girder G1 & interior girder G2 in kip-ft
bm1 = df['G1 - max_bm']/(1000*12)
bm2 = df['G2 - max_bm']/(1000*12)
bm3 = df['G3 - max_bm']/(1000*12)
bm4 = df['G4 - max_bm']/(1000*12)

# reaction in girders G1 & G2 in kip. (only A1 taken due to symmetry)
r1 = df['G1-A1-Y']/1000
```

```
r2 = df['G2-A1-Y']/1000
r3 = df['G3-A1-Y']/1000
r4 = df['G4-A1-Y']/1000

# line analysis with full railing load assumed to be applied on a girder
bm_line = q_rail * (L - 2*9/12)**2 / 8
r_line = q_rail * L / 2


# normalizing bending moments with respect to line analysis
n_bm1 = bm1 / bm_line
n_bm2 = bm2 / bm_line
n_bm3 = bm3 / bm_line
n_bm4 = bm4 / bm_line

# normalizing vertical reactions with respect to line analysis
n_r1 = r1 / r_line
n_r2 = r2 / r_line
n_r3 = r3 / r_line
n_r4 = r4 / r_line

# add the distribution factors to the dataframe
df['n_bm1'] = n_bm1
df['n_bm2'] = n_bm2
df['n_bm3'] = n_bm3
df['n_bm4'] = n_bm4
df['n_r1'] = n_r1
df['n_r2'] = n_r2
df['n_r3'] = n_r3
df['n_r4'] = n_r4

# sample data from leve rule
# n_r1 = (w_oh + S) / S

X = np.stack((L, w_oh, S), axis=-1)
y = n_bm2

# Learn equations
default_pysr_params = dict(
    populations=30,
    model_selection="best",
)

# model = PySRRegressor(
#     niterations=30,
#     binary_operators=['+', '-', '*', '/', '^'],
#     unary_operators=["square", "cube", "sqrt"],
#     **default_pysr_params
# )
```

```
model = PySRRegressor(
    niterations=50,  # < Increase me for better results
    binary_operators=['+', '-', '*', '/', '^', "physics(x, y) = x^2 / y"],
    unary_operators=["square", "cube", "sqrt", "inv(x) = 1/x"],
        # ^ Custom operator (julia syntax)
    extra_sympy_mappings={"inv": lambda x: 1 / x,
                          "physics": lambda x, y: x**2 / y},
    # ^ Define operator for SymPy as well
    loss="loss(prediction, target) = (prediction - target)^2",
    # ^ Custom loss function (julia syntax)
    **default_pysr_params
)

model.fit(X, y)

print(model)
```

```
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:1346: UserWarning: Note: it looks
  warnings.warn(
Compiling Julia backend...
/usr/local/lib/python3.10/dist-packages/pysr/julia_helpers.py:231: UserWarning: Julia
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:109: UserWarning: You are using th
  warnings.warn(
WARNING: using StaticArrays.setindex in module FiniteDiff conflicts with an existing
Started!

Expressions evaluated per second: 4.700e+03
Head worker occupation: 9.6%
Progress: 14 / 1500 total iterations (0.933%)
=====================================================================================
Hall of Fame:
-------------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           2.951e-03  1.594e+01  y = 0.3203
4           2.096e-03  1.141e-01  y = (square(0.59217) ^ 1.187)
5           1.521e-03  3.205e-01  y = sqrt(inv(x₂ + x₁))
7           1.410e-03  3.769e-02  y = inv(sqrt(x₂ + (x₁ ^ 1.3337)))
9           1.404e-03  2.169e-03  y = sqrt(inv((physics(2.4004, x₂) + x₂) + x₁))
11          1.398e-03  2.136e-03  y = sqrt(inv((physics(1.9246, x₂ - x₁) + x₂) + x₁))
14          1.168e-03  5.989e-02  y = square(0.59217 + (physics(-0.45884, x₀ * ((0.04
                                  46) / x₂)) / -1.7503))
15          8.867e-04  2.759e-01  y = square(0.59217 + (physics(-0.45884, square(x₀ *
                                  ^ 0.48246) / x₂))) / -1.7503))
17          7.518e-04  8.247e-02  y = square(0.59217 + (physics(-0.45884, square(x₀ *
                                  ^ 0.48246) / x₂))) / (-1.7503 * 0.84552)))
-------------------------------------------------------------------------------------
=====================================================================================
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 1.900e+04
Head worker occupation: 3.3%
Progress: 84 / 1500 total iterations (5.600%)
=====================================================================================
```

```
Hall of Fame:
-----------------------------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           2.088e-03  1.594e+01  y = 0.29093
5           1.521e-03  7.928e-02  y = sqrt(inv(x₂ + x₁))
7           1.410e-03  3.769e-02  y = inv(sqrt(x₂ + (x₁ ^ 1.3337)))
8           1.404e-03  4.257e-03  y = sqrt(inv(x₂ + (x₁ / cube(0.90602))))
9           1.332e-03  5.327e-02  y = sqrt(inv((x₂ + x₁) + physics(x₂, x₀)))
11          9.700e-04  1.584e-01  y = (x₀ / (((x₀ + x₂) / 0.39173) + (x₂ / 0.19911)))
12          8.022e-04  1.900e-01  y = (x₀ / (((x₀ + square(x₁)) / 0.39173) + (x₂ / 0.
15          7.469e-04  2.379e-02  y = (x₀ / ((x₀ / 0.39173) + (x₂ * (-1.4959 * (-0.86
                                   x₁))))))
17          6.660e-04  5.736e-02  y = (x₀ / ((x₀ / 0.39173) + (x₂ * (-1.4959 * (-0.86
                                  + 0.63926) + x₁))))))
-----------------------------------------------------------------------------------------------------
=====================================================================================================
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 2.530e+04
Head worker occupation: 2.5%
Progress: 157 / 1500 total iterations (10.467%)
```

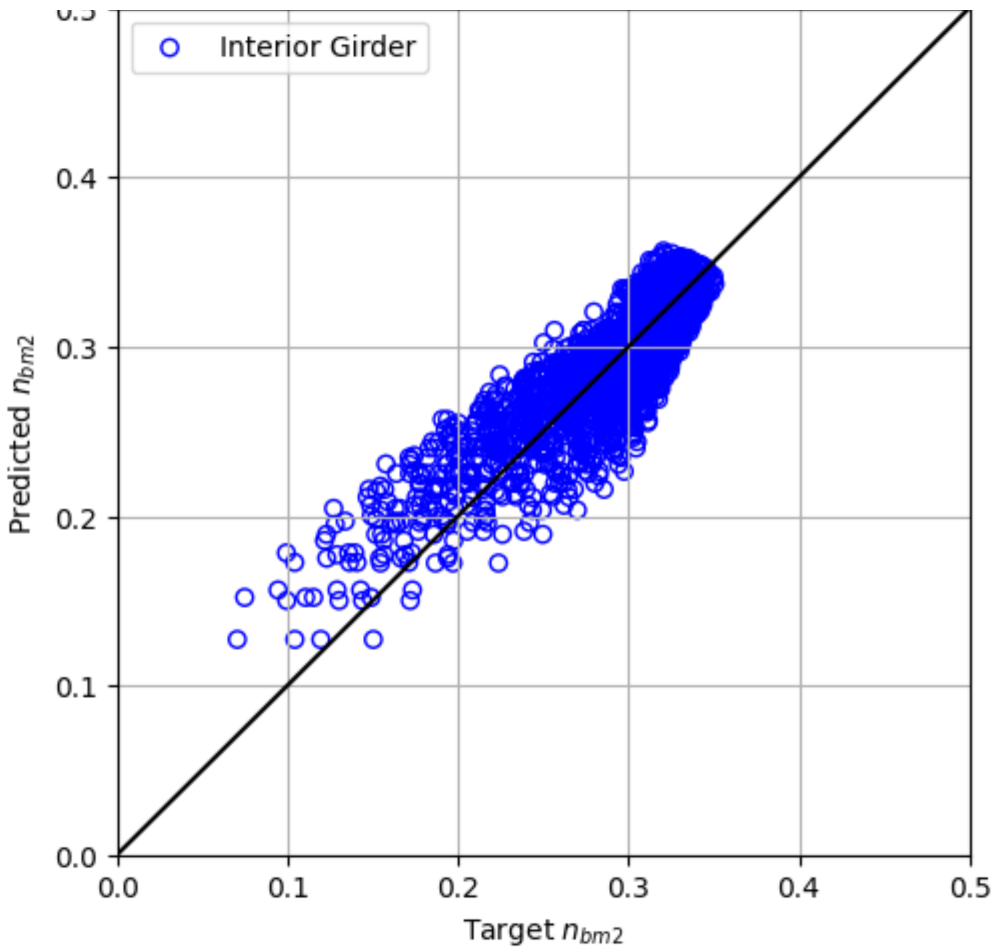
```
model.sympy()
```

$$\frac{0.37524402\left(x_0 - x_1\left(-x_1 + x_2\right)\right)}{x_0}$$

```
n_bm2_pred = model.predict(X)
ypredict_simpler = model.predict(X, 2)

print("Default selection MSE:", np.power(n_bm2_pred - y, 2).mean())
print("Manual selection MSE for index 2:", np.power(ypredict_simpler - y, 2).mean())
```

```
Default selection MSE: 0.1596117656019573
Manual selection MSE for index 2: 0.155651053727596
```

```
x_line = [0, 0.5]
y_line = [0, 0.5]

fig, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(5, 5), constrained_layout = True)

ax1.scatter(x=n_bm2, y=n_bm2_pred, marker='o', c='none', edgecolor='b', label='Interior G
ax1.plot(x_line, y_line, c = "k")
ax1.set_title('Accuracy')
ax1.legend()
plt.xlim((0,0.5))
plt.ylim((0,0.5))
ax1.set_xlabel('Target $n_{bm2}$')
ax1.set_ylabel('Predicted $n_{bm2}$')
ax1.grid()
```




```
model.latex()
```

```
'\\frac{0.375 \\left(x_{0} - x_{1} \\left(- x_{1} + x_{2}\\right)\\right)}{x_{0}}'
```

The following expression best suits the data for n_{bm2}

$$\frac{0.375(L - w_{oh}(-w_{oh} + S))}{L}$$

```
X = np.stack((L, w_oh, S), axis=-1)
y = n_bm1

# Learn equations
default_pysr_params = dict(
    populations=30,
    model_selection="best",
)

# model = PySRRegressor(
#     niterations=30,
#     binary_operators=['+', '-', '*', '/', '^'],
#     unary_operators=["square", "cube", "sqrt"],
#     **default_pysr_params
# )
```

```
# )

model1 = PySRRegressor(
    niterations=50,  # < Increase me for better results
    binary_operators=['+', '-', '*', '/', '^', "physics(x, y) = x^2 / y"],
    unary_operators=["square", "cube", "sqrt", "inv(x) = 1/x"],
        # ^ Custom operator (julia syntax)
    extra_sympy_mappings={"inv": lambda x: 1 / x,
                          "physics": lambda x, y: x**2 / y},
    # ^ Define operator for SymPy as well
    loss="loss(prediction, target) = (prediction - target)^2",
    # ^ Custom loss function (julia syntax)
    **default_pysr_params
)

model1.fit(X, y)

print(model1)
```

```
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:1346: UserWarning: Note: it looks
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pysr/julia_helpers.py:231: UserWarning: Julia
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:109: UserWarning: You are using th
  warnings.warn(
Started!

Expressions evaluated per second: 3.980e+03
Head worker occupation: 0.2%
Progress: 9 / 1500 total iterations (0.600%)
=====================================================================================
Hall of Fame:
-------------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           2.216e-02  1.594e+01  y = 0.56201
2           1.915e-02  1.462e-01  y = sqrt(0.56201)
5           1.182e-02  1.609e-01  y = (0.43862 ^ (0.43862 / 0.86458))
8           1.179e-02  6.846e-04  y = ((0.9967 + -0.32363) / square(sqrt(square(1.008
9           1.165e-02  1.258e-02  y = (((sqrt(x₁) ^ inv(x₀)) + -0.32363) / 1.0089)
10          6.842e-03  5.319e-01  y = sqrt(sqrt(inv(inv(sqrt(x₂))) + -0.19604) + -1.1
16          5.679e-03  3.105e-02  y = sqrt(x₁ / sqrt(sqrt(square((x₂ * 1.2612) / phys
                                  ) * square(x₁))))
18          5.371e-03  2.788e-02  y = sqrt(x₁ / sqrt(sqrt(square((x₂ * 1.2612) / phys
                                  ) * square(x₁)) + 1.1501))
20          4.889e-03  4.704e-02  y = sqrt(x₁ / sqrt(sqrt(square((x₂ * 1.3683) / phys
                                  - (0.90887 - x₂))) * square(x₁))))
-------------------------------------------------------------------------------------
=====================================================================================
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 1.320e+04
Head worker occupation: 0.4%
Progress: 65 / 1500 total iterations (4.333%)
=====================================================================================
```

```
Hall of Fame:
-----------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           1.179e-02  1.594e+01  y = 0.66388
2           1.179e-02  0.000e+00  y = sqrt(0.44072)
5           1.003e-02  5.385e-02  y = physics(0.76532, 0.97984 ^ x₂)
7           5.270e-03  3.217e-01  y = sqrt(0.44509 ^ (sqrt(x₀) / x₂))
9           5.257e-03  1.228e-03  y = sqrt(sqrt(physics(-0.41031, square(sqrt(x₀) / x
11          5.164e-03  8.943e-03  y = sqrt(sqrt(physics(-0.46554, square((sqrt(x₀) +
                                  ₂))))
12          4.671e-03  1.004e-01  y = sqrt(sqrt(sqrt(x₁) / sqrt(square(x₂ / physics(x
14          4.044e-03  7.208e-02  y = sqrt(sqrt(physics(0.55365 / -1.4548, physics(sq
                                  4851) / x₂, x₁))))
-----------------------------------------------------------------------------------
===================================================================================
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 1.400e+04
Head worker occupation: 0.7%
Progress: 96 / 1500 total iterations (6.400%)
===================================================================================
Hall of Fame:
-----------------------------------------------------------------------------------
```

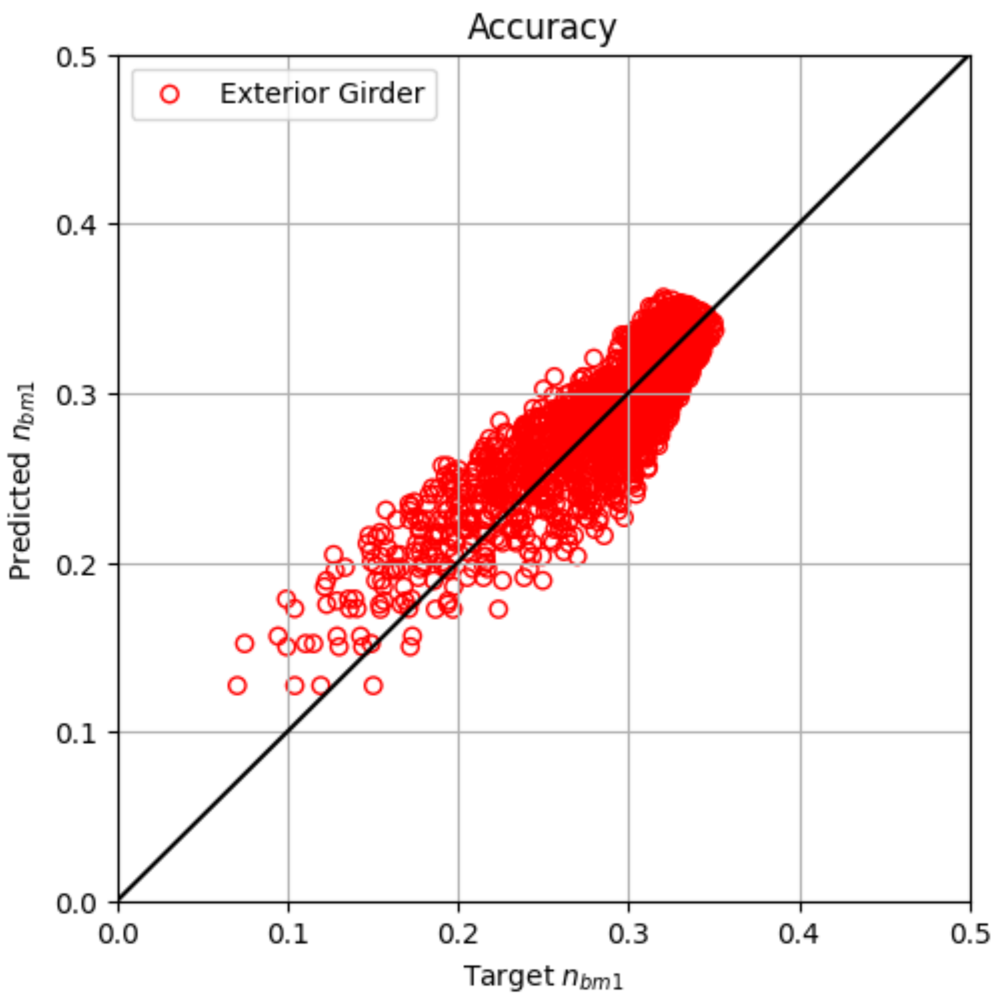
```
model1.sympy()
```

$$\left(\frac{x_0}{x_1}\right)^{-\frac{1.0396062}{x_2}}$$

```
n_bm1_pred = model1.predict(X)
ypredict_simpler = model1.predict(X, 2)

print("Default selection MSE:", np.power(n_bm1_pred - y, 2).mean())
print("Manual selection MSE for index 2:", np.power(ypredict_simpler - y, 2).mean())
```

```
Default selection MSE: 0.0043169609843600465
Manual selection MSE for index 2: 0.00676906220853474
```

```
x_line = [0, 0.5]
y_line = [0, 0.5]

fig, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(5, 5), constrained_layout = True)

ax1.scatter(x=n_bm2, y=n_bm2_pred, marker='o', c='none', edgecolor='r', label='Exterior G
ax1.plot(x_line, y_line, c = "k")
ax1.set_title('Accuracy')
ax1.legend()
plt.xlim((0,0.5))
plt.ylim((0,0.5))
ax1.set_xlabel('Target $n_{bm1}$')
ax1.set_ylabel('Predicted $n_{bm1}$')
ax1.grid()
```




```
model1.latex()
```

```
'\\left(\\frac{x_{0}}{x_{1}}\\right)^{- \\frac{1.04}{x_{2}}}'
```

The following expression best suits the data for n_{bm1}

$$\left(\frac{L}{w_{oh}}\right)^{- \frac{1.04}{S}}$$

Symbolic Regression for Reactions

```
%%shell
set -e

#---------------------------------------------------#
JULIA_VERSION="1.8.5"
export JULIA_PKG_PRECOMPILE_AUTO=0
#---------------------------------------------------#

if [ -z `which julia` ]; then
  # Install Julia
  JULIA_VER=`cut -d '.' -f -2 <<< "$JULIA_VERSION"`
  echo "Installing Julia $JULIA_VERSION on the current Colab Runtime..."
  BASE_URL="https://julialang-s3.julialang.org/bin/linux/x64"
  URL="$BASE_URL/$JULIA_VER/julia-$JULIA_VERSION-linux-x86_64.tar.gz"
  wget -nv $URL -O /tmp/julia.tar.gz # -nv means "not verbose"
  tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
  rm /tmp/julia.tar.gz

  echo "Installing PyCall.jl..."
  julia -e 'using Pkg; Pkg.add("PyCall"); Pkg.build("PyCall")'
  julia -e 'println("Success")'

fi
```

```
Installing Julia 1.8.5 on the current Colab Runtime...
2023-12-04 03:34:56 URL:https://storage.googleapis.com/julialang2/bin/linux/x64/1.8/j
Installing PyCall.jl...
  Installing known registries into `~/.julia`
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
   Installed Conda ─────────── v1.10.0
   Installed PyCall ────────── v1.96.2
   Installed VersionParsing ── v1.3.0
   Installed JSON ──────────── v0.21.4
   Installed Parsers ───────── v2.8.0
   Installed MacroTools ────── v0.5.11
   Installed Preferences ───── v1.4.1
   Installed PrecompileTools ─ v1.2.0
    Updating `~/.julia/environments/v1.8/Project.toml`
  [438e738f] + PyCall v1.96.2
    Updating `~/.julia/environments/v1.8/Manifest.toml`
  [8f4d0f93] + Conda v1.10.0
  [682c06a0] + JSON v0.21.4
  [1914dd2f] + MacroTools v0.5.11
  [69de0a69] + Parsers v2.8.0
  [aea7be01] + PrecompileTools v1.2.0
  [21216c6a] + Preferences v1.4.1
```

```
  [438e738f] + PyCall v1.96.2
  [81def892] + VersionParsing v1.3.0
  [0dad84c5] + ArgTools v1.1.1
  [56f22d72] + Artifacts
  [2a0f44e3] + Base64
  [ade2ca70] + Dates
  [f43a241f] + Downloads v1.6.0
  [7b1f6079] + FileWatching
  [b27032c2] + LibCURL v0.6.3
  [8f399da3] + Libdl
  [37e2e46d] + LinearAlgebra
  [d6f4376e] + Markdown
  [a63ad114] + Mmap
  [ca575930] + NetworkOptions v1.2.0
  [de0858da] + Printf
  [9a3f8284] + Random
  [ea8e919c] + SHA v0.7.0
  [9e88b42a] + Serialization
  [fa267f1f] + TOML v1.0.0
  [cf7118a7] + UUIDs
  [4ec0a83e] + Unicode
  [e66e0078] + CompilerSupportLibraries_jll v1.0.1+0
  [deac9b47] + LibCURL_jll v7.84.0+0
  [29816b5a] + LibSSH2_jll v1.10.2+0
  [c8ffd9c3] + MbedTLS_jll v2.28.0+0
  [14a3606d] + MozillaCACerts_jll v2022.2.1
  [4536629a] + OpenBLAS_jll v0.3.20+0
  [83775a58] + Zlib_jll v1.2.12+3
  [8e850b90] + libblastrampoline_jll v5.1.1+0
  [8e850ede] + nghttp2_jll v1.48.0+0
    Building Conda ─→ `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/51
    Building PyCall → `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/1c
    Building Conda ─→ `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/51
    Building PyCall → `~/.julia/scratchspaces/44cfe95a-1eb2-52ea-b672-e2afdf69b78f/1c
Success
```

Install PySR and PyTorch-Lightning:

```
%pip install -Uq pysr pytorch_lightning --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 2.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 776.9/776.9 kB 24.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 68.7/68.7 kB 10.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 806.1/806.1 kB 63.1 MB/s eta 0:00:00
```

```
from julia import Julia

julia = Julia(compiled_modules=False, threads="auto")
from julia import Main
from julia.tools import redirect_output_streams

redirect_output_streams()
```

```
import pysr

# We don't precompile in colab because compiled modules are incompatible static Python li
pysr.install(precompile=False)
```

```
Julia Version 1.8.5
Commit 17cfb8e65ea (2023-01-08 06:45 UTC)
Platform Info:
  OS: Linux (x86_64-linux-gnu)
      Ubuntu 22.04.3 LTS
  uname: Linux 5.15.120+ #1 SMP Wed Aug 30 11:19:59 UTC 2023 x86_64 x86_64
  CPU: Intel(R) Xeon(R) CPU @ 2.00GHz:
              speed         user         nice          sys         idle          irq
       #1  2000 MHz       1016 s          0 s        168 s       1130 s          0 s
       #2  2000 MHz       1100 s          0 s        166 s       1054 s          0 s
  Memory: 12.6783447265625 GB (11530.61328125 MB free)
  Uptime: 240.41 sec
  Load Avg:  2.83  1.17  0.46
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-13.0.1 (ORCJIT, skylake-avx512)
  Threads: 1 on 2 virtual cores
Environment:
  LD_LIBRARY_PATH = /usr/lib64-nvidia
  JULIA_PROJECT = @pysr-0.16.3
  JULIA_PKG_PRECOMPILE_AUTO = 0
  TCLLIBPATH = /usr/share/tcltk/tcllib1.20
  HOME = /root
  PYTHONPATH = /env/python
  LIBRARY_PATH = /usr/local/cuda/lib64/stubs
  PATH = /opt/bin:/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/loca
  COLAB_DEBUG_ADAPTER_MUX_PATH = /usr/local/bin/dap_multiplexer
  TERM = xterm-color
Julia Version 1.8.5
Commit 17cfb8e65ea (2023-01-08 06:45 UTC)
Platform Info:
  OS: Linux (x86_64-linux-gnu)
      Ubuntu 22.04.3 LTS
  uname: Linux 5.15.120+ #1 SMP Wed Aug 30 11:19:59 UTC 2023 x86_64 x86_64
  CPU: Intel(R) Xeon(R) CPU @ 2.00GHz:
              speed         user         nice          sys         idle          irq
       #1  2000 MHz       1345 s          0 s        192 s       1184 s          0 s
       #2  2000 MHz       1432 s          0 s        192 s       1101 s          0 s
  Memory: 12.6783447265625 GB (10983.30859375 MB free)
  Uptime: 281.1 sec
  Load Avg:  2.76  1.38  0.56
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-13.0.1 (ORCJIT, skylake-avx512)
  Threads: 1 on 2 virtual cores
Environment:
  LD_LIBRARY_PATH = /usr/lib64-nvidia
  JULIA_PROJECT = @pysr-0.16.3
  JULIA_PKG_PRECOMPILE_AUTO = 0
  TCLLIBPATH = /usr/share/tcltk/tcllib1.20
```

```
        TCLLIBPATH = /usr/share/tcltk/tcllib1.20
        HOME = /root
        PYTHONPATH = /env/python
        LIBRARY_PATH = /usr/local/cuda/lib64/stubs
        PATH = /opt/bin:/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/loca
        COLAB_DEBUG_ADAPTER_MUX_PATH = /usr/local/bin/dap_multiplexer
        TERM = xterm-color
      [ Info: Julia version info
      [ Info: Julia executable: /usr/local/bin/julia
```

```
import sympy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from pysr import PySRRegressor
from sklearn.model_selection import train_test_split

df = pd.read_csv('https://raw.githubusercontent.com/bhushanrajs/sciml_project/main/analys

tx_girders = {'Tx28' : {'D' : 28.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx34' : {'D' : 34.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx40' : {'D' : 40.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx46' : {'D' : 46.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx54' : {'D' : 54.0, 'b1' : 36.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx62' : {'D' : 62.0, 'b1' : 42.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx70' : {'D' : 70.0, 'b1' : 42.0, 'b2' : 7.0, 'b3' : 32.0, 'b4' : 2.0, 'b5
              'Tx84' : {'D' : 84.0, 'b1' : 58.0, 'b2' : 8.0, 'b3' : 38.0, 'b4' : 3.0, 'b5
              }

# bridge geometry data
L = df['L']/12 # span length in ft.
S = df['S']/12 # girder spacing in ft.
w_oh = df['w_oh']/12 # overhang width in ft.
ts = df['ts_U']/12 # thickness of overhang in ft.
girder = df['Girder Type']
D = []
for _, girder_type in girder.items():
  D.append(tx_girders[girder_type]['D'])
df['D'] = D

# intensity of railing dead load in kip/ft
b_rail = df['b_rail_left'] # width of railing
q_rail = (df['q_rail_left'] * b_rail * 12)/1000

# max bending moment in exterior girder G1 & interior girder G2 in kip-ft
bm1 = df['G1 - max_bm']/(1000*12)
bm2 = df['G2 - max_bm']/(1000*12)
bm3 = df['G3 - max_bm']/(1000*12)
bm4 = df['G4 - max_bm']/(1000*12)

# reaction in girders G1 & G2 in kip. (only A1 taken due to symmetry)
```

```
r1 = df['G1-A1-Y']/1000
r2 = df['G2-A1-Y']/1000
r3 = df['G3-A1-Y']/1000
r4 = df['G4-A1-Y']/1000

# line analysis with full railing load assumed to be applied on a girder
bm_line = q_rail * (L - 2*9/12)**2 / 8
r_line = q_rail * L / 2


# normalizing bending moments with respect to line analysis
n_bm1 = bm1 / bm_line
n_bm2 = bm2 / bm_line
n_bm3 = bm3 / bm_line
n_bm4 = bm4 / bm_line

# normalizing vertical reactions with respect to line analysis
n_r1 = r1 / r_line
n_r2 = r2 / r_line
n_r3 = r3 / r_line
n_r4 = r4 / r_line

# add the distribution factors to the dataframe
df['n_bm1'] = n_bm1
df['n_bm2'] = n_bm2
df['n_bm3'] = n_bm3
df['n_bm4'] = n_bm4
df['n_r1'] = n_r1
df['n_r2'] = n_r2
df['n_r3'] = n_r3
df['n_r4'] = n_r4

# sample data from leve rule
# n_r1 = (w_oh + S) / S

X = np.stack((L, S, w_oh), axis=-1)
y = n_r1

# Learn equations
default_pysr_params = dict(
    populations=50,
    model_selection="best",
)

# model = PySRRegressor(
#     niterations=30,
#     binary_operators=['+', '-', '*', '/', '^'],
#     unary_operators=["square", "cube", "sqrt"],
#     **default_pysr_params
# )
```

```
model = PySRRegressor(
    niterations=40,  # < Increase me for better results
    binary_operators=['+', '-', '*', '/', '^', "physics(x, y) = x^2 / y"],
    unary_operators=["square", "cube", "sqrt", "inv(x) = 1/x"],
        # ^ Custom operator (julia syntax)
    extra_sympy_mappings={"inv": lambda x: 1 / x,
                          "physics": lambda x, y: x**2 / y},
    # ^ Define operator for SymPy as well
    loss="loss(prediction, target) = (prediction - target)^2",
    # ^ Custom loss function (julia syntax)
    **default_pysr_params
)

model.fit(X, y)

print(model)
```

```
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:1346: UserWarning: Note: it looks
  warnings.warn(
Compiling Julia backend...
/usr/local/lib/python3.10/dist-packages/pysr/julia_helpers.py:231: UserWarning: Julia
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:109: UserWarning: You are using th
  warnings.warn(
WARNING: using StaticArrays.setindex in module FiniteDiff conflicts with an existing
Started!

Expressions evaluated per second: 3.700e+02
Head worker occupation: 13.5%
Progress: 5 / 2000 total iterations (0.250%)
=====================================================================================
Hall of Fame:
-------------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           1.596e-01  1.594e+01  y = 1.3766
2           6.970e-02  8.286e-01  y = sqrt(0.53628)
3           6.252e-03  2.411e+00  y = (x₀ / x₀)
5           6.033e-03  1.784e-02  y = sqrt(1.3766 ^ cube(-0.49618))
6           5.432e-03  1.050e-01  y = (1.3766 ^ (cube(-0.49618) / x₂))
7           5.226e-03  3.858e-02  y = square(1.3766 ^ (cube(-0.49618) / x₂))
8           5.052e-03  3.389e-02  y = (1.3766 ^ (-0.49618 / physics(square(x₂), 0.974
9           3.699e-03  3.116e-01  y = cube(1.3766 ^ (-0.49618 / physics(square(x₂), 1
11          3.658e-03  5.538e-03  y = inv(x₂ ^ ((0.65418 ^ (x₂ * square(x₂))) ^ 0.654
12          3.421e-03  6.705e-02  y = inv(sqrt(square(x₂) ^ (0.65418 ^ ((x₂ ^ x₂) + x
15          3.410e-03  1.127e-03  y = (0.53628 ^ (physics(0.41827 ^ x₂, inv(cube(0.96
                                  2133 / -0.048011))) / x₁))
16          3.394e-03  4.718e-03  y = sqrt(inv(x₂ ^ ((0.65418 ^ ((sqrt(x₂) ^ x₂) * sq
                                  ^ square(0.65418))))
17          2.836e-03  1.796e-01  y = square(cube(cube(((x₀ - (0.79069 ^ square(x₂)))
                                  0.79069, x₁ ^ 0.40763)) / x₀)))
20          2.448e-03  4.904e-02  y = square(cube(cube(((x₀ - (sqrt(sqrt(0.40763)) ^
                                  ) + physics(0.69369, x₀ ^ square(0.38578))) / x₀)))
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 7.730e+03
Head worker occupation: 0.6%
Progress: 34 / 2000 total iterations (1.700%)
======================================================================================
Hall of Fame:
--------------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           6.017e-03  1.594e+01  y = 0.98466
5           2.465e-03  2.231e-01  y = (physics(1.561, x₂) ^ -0.1962)
6           2.394e-03  2.931e-02  y = sqrt(sqrt(inv(physics(1.6551, x₂))))
7           2.200e-03  8.453e-02  y = (physics(1.561, x₂ - 0.15187) ^ -0.1962)
9           2.191e-03  2.073e-03  y = (physics(1.561, x₂ - physics(0.52146, 1.5432))
11          2.090e-03  2.348e-02  y = (physics(1.561, x₂ - physics(1.561 - 0.17316, x
                                  62)
13          1.253e-03  2.557e-01  y = sqrt(sqrt(sqrt(sqrt((square(sqrt(x₁)) / x₀) ^ 1
                                  ₂))
15          1.188e-03  2.678e-02  y = sqrt(sqrt(sqrt(sqrt((square(sqrt(x₁)) / x₀) ^ (
                                  0886))) * x₂))
17          1.120e-03  2.929e-02  y = sqrt(sqrt(sqrt(sqrt(((square(sqrt(x₁)) / x₀) ^
```

```
model.sympy()
```

```
n_r1_pred = model.predict(X)
# n_r1_pred = 0.757 + w_oh * D**0.5/L
print("Default selection MSE:", np.power(n_r1_pred - y, 2).mean())
```

```
Default selection MSE: 0.0008493762544633727
```

```
x_line = [0.5, 1.5]
y_line = [0.5, 1.5]

fig, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(5, 5), constrained_layout = True)

ax1.scatter(x=n_r1, y=n_r1_pred, marker='o', c='none', edgecolor='r', label='Exterior Gir
ax1.plot(x_line, y_line, c = "k")
ax1.set_title('Accuracy')
ax1.legend()
plt.xlim((0.7,1.3))
plt.ylim((0.7,1.3))
ax1.set_xlabel('Target $n_{r1}$')
ax1.set_ylabel('Predicted $n_{r1}$')
ax1.grid()
```

```
model.latex()
```

Among all the solutions given by the model. We found the following equations best suited for our data.

1. $0.881 + \frac{1.77S^2 w_{oh}}{L^2}$
2. $1.32\left(\frac{w_{oh}^2}{L}\right)^{0.117}$

However, the third equations seems to be more simpler and suited for our data.

```
y2 = n_r2
model2 = PySRRegressor(
    niterations=40,  # < Increase me for better results
    binary_operators=['+', '-', '*', '/', '^', "physics(x, y) = x^2 / y"],
    unary_operators=["square", "cube", "sqrt", "inv(x) = 1/x"],
        # ^ Custom operator (julia syntax)
    extra_sympy_mappings={"inv": lambda x: 1 / x,
                          "physics": lambda x, y: x**2 / y},
    # ^ Define operator for SymPy as well
    loss="loss(prediction, target) = (prediction - target)^2",
```

```
    # ^ Custom loss function (julia syntax)
    **default_pysr_params
)

model2.fit(X, y2)

print(model)
```

```
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:1346: UserWarning: Note: it looks
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pysr/julia_helpers.py:231: UserWarning: Julia
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/pysr/sr.py:109: UserWarning: You are using th
  warnings.warn(
Started!

Expressions evaluated per second: 3.300e+03
Head worker occupation: 0.1%
Progress: 7 / 2000 total iterations (0.350%)
==================================================================================
Hall of Fame:
----------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           4.506e-03  1.594e+01  y = 0.036309
3           3.172e-03  1.755e-01  y = physics(0.3492, x₂)
5           2.195e-03  1.841e-01  y = cube(sqrt(0.59158) / x₂)
10          2.076e-03  1.113e-02  y = ((square(-0.18677 / sqrt(x₂)) * sqrt(x₀)) / x₂)
11          2.059e-03  8.267e-03  y = inv(((x₂ ^ x₂) + 1.2765) + square(x₂ * -1.5983)
12          2.037e-03  1.079e-02  y = ((square(-0.18677 / sqrt(cube(sqrt(x₂)))) * sqr
                                  )
14          1.903e-03  3.401e-02  y = ((square(-0.18677 / sqrt(x₂)) * (square(sqrt(sq
                                  x₂)) / x₂)
16          1.800e-03  2.775e-02  y = ((square(-0.18677 / sqrt(cube(sqrt(x₂ / 1.2927)
                                  (x₀) - x₂)) / x₂)
----------------------------------------------------------------------------------
==================================================================================
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 3.200e+03
Head worker occupation: 0.4%
Progress: 20 / 2000 total iterations (1.000%)
==================================================================================
Hall of Fame:
----------------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           4.506e-03  1.594e+01  y = 0.036309
3           3.172e-03  1.755e-01  y = physics(0.3492, x₂)
4           2.216e-03  3.585e-01  y = (0.47802 / cube(x₂))
5           2.195e-03  9.567e-03  y = cube(sqrt(0.59158) / x₂)
9           2.166e-03  3.300e-03  y = inv(((x₂ + x₂) * square(x₂)) + 1.0957)
10          2.054e-03  5.320e-02  y = (inv(x₂ + physics(cube(x₂), x₀)) * square(-0.39
12          1.955e-03  2.473e-02  y = sqrt(physics(square(0.097763), cube(cube(sqrt(x
                                  )))) * x₀)
13          1.844e-03  5.867e-02  y = (inv(x₂ + physics(cube(cube(x₂ + -1.153)), x₀))
```

```
                                    0.39698))
15          1.810e-03  9.289e-03  y = (inv(sqrt(square(x₂ + physics(square(cube(x₂ +
                                  ₀)))) * square(-0.39698))
16          1.674e-03  7.787e-02  y = physics(cube(0.87605 / (0.99768 ^ (x₀ - (x₂ ^ x
                                  39678, cube(sqrt(x₂)))
20          1.518e-03  2.452e-02  y = physics(cube(sqrt(square(0.87605)) / (0.99768 ^
                                  e(sqrt(x₂)) ^ x₂)))) * -0.39678, square(sqrt(x₂)))
---------------------------------------------------------------------------------------
=======================================================================================
Press 'q' and then <enter> to stop execution early.

Expressions evaluated per second: 1.250e+04
```

```
model2.sympy()
```

```
n_r2_pred = model2.predict(X)

# n_r2_pred =  0.149 - 1.09 * w_oh**2/L
print("Default selection MSE:", np.power(n_r2_pred - y2, 2).mean())
```

```
Default selection MSE: 0.0005198363132888174
```

```
x_line = [-0.25, 0.25]
y_line = [-0.25, 0.25]

fig, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(5, 5), constrained_layout = True)

ax1.scatter(x=n_r2, y=n_r2_pred, marker='o', c='none', edgecolor='b', label='Interior Gir
ax1.plot(x_line, y_line, c = "k")
ax1.set_title('Accuracy')
ax1.legend()
plt.xlim((-0.25, 0.25))
plt.ylim((-0.25, 0.25))
ax1.set_xlabel('Target $n_{r2}$')
ax1.set_ylabel('Predicted $n_{r2}$')
ax1.grid()
```

```
model2.latex()
```

Among all the solutions given by the model. We found the following equations best suited for our data.

1. $\frac{w_{oh}^2}{L - \frac{D}{w_{oh}}} + 0.167$
2. $0.149 - \frac{1.09 w_{oh}^2}{L}$
3. $0.14 - \frac{w_{oh}^2}{L}$

For the interior girder, we chose the third equation as the best.