# GreenLearning Documentation

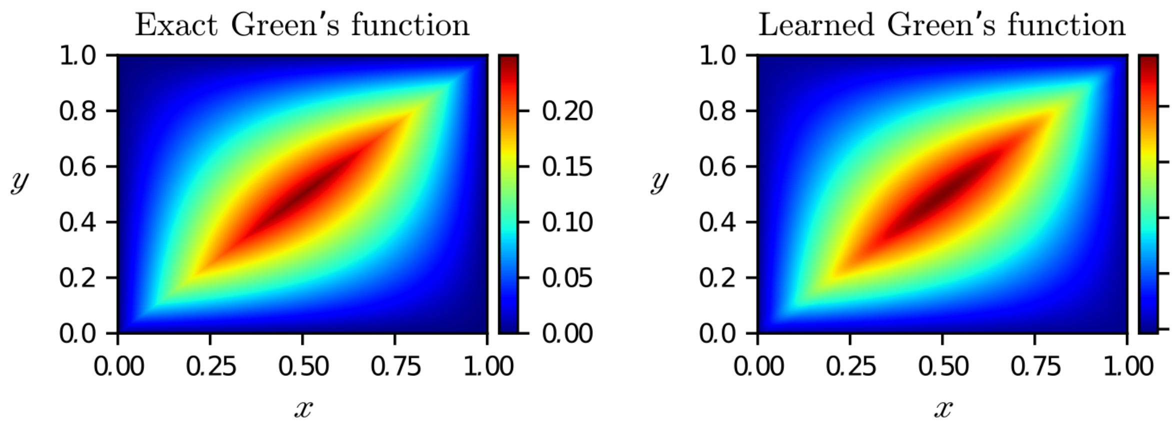
***Release 1.0.0***

**Nicolas Boulle**

**May 04, 2021**

# CONTENTS

GreenLearning is a deep learning library based on Tensorflow for learning Green's functions associated with partial differential operators.



*Exact and learned Green's function of the Laplace operator.*

The library is maintained by Nicolas Boullé. If you are interested in using it, do not hesitate to get in contact at `boulle@maths.ox.ac.uk`.

CHAPTER

ONE

# FEATURES

- GreenLearning learns Green's functions and homogeneous solutions associated with scalar and systems of linearized partial differential equations in 1D and 2D with deep learning.
- Rational neural networks are implemented and used to increase the accuracy of the learned Green's functions.
- GreenLearning requires no hyperparameter tuning to successfully learn Green's functions.
- The neural networks can be created and trained easily with a few lines of code.
- It is simple to generate the training datasets with MATLAB scripts.

# CHAPTER TWO

# GUIDE

See the following sections to learn how to install and use the GreenLearning library.

## 2.1 Installation

### 2.1.1 Requirements

GreenLearning relies on the following Python libraries:

- TensorFlow>=1.15.0
- Matplotlib
- NumPy
- SciPy

### 2.1.2 How to install GreenLearning

- For users, you can install the stable version with `pip`:

  ```
  pip install greenlearning
  ```

  or with `conda`:

  ```
  conda install -c conda-forge greenlearning
  ```

- For developers, you should clone the GitHub repository and install it manually on your machine:

  ```
  git clone https://github.com/NBoulle/greenlearning.git
  cd greenlearning
  pip install -e.
  ```

## 2.2 Guide

### 2.2.1 Creating a training dataset

In this section, we explain how to generate a training dataset in MATLAB to learn the Green's function associated to an ODE or a system of ODEs.

This step requires the MATLAB package called Chebfun (see https://www.chebfun.org/download/ for installation instructions).

#### Definition of the differential operator

The definition of the differential operator can be done by creating a MATLAB script in a folder `examples/`. In the following example, we add a script called `helmholtz.m` in the folder with the following content:

```
function output_example = helmholtz()
% Helmholtz equation

% Define the domain.
dom = [0,1];

% Parameter of the equation
K = 15;

% Differential operator
N = chebop(@(x,u) diff(u,2)+K^2*u, dom);

% Boundary conditions
N.bc = @(x,u) [u(dom(1)); u(dom(2))];

% Output
output_example = {N};
end
```

Here, we define a Helmholtz operator $\mathcal{L}u = \frac{d^2u}{dx^2} + K^2u$ with Helmholtz frequency $K = 15$, on the interval $[0, 1]$, with homogeneous Dirichlet boundary conditions.

- One could, for instance, define the boundary conditions to be $u(0) = -1$, $u(1) = 2$ as follows:

  ```
  % Boundary conditions
  N.bc = @(x,u) [u(dom(1))+1; u(dom(2))-2];
  ```

- It is also possible to impose $u(0) = 0$, $u'(0) = 1$ with the following line:

  ```
  % Boundary conditions
  N.bc = @(x,u) [u(dom(1)); feval(diff(u),dom(1))-1];
  ```

If the exact Green's function is known, one can in addition provide its expression (as a `string` in `numpy` format) to compare with the learned Green's function:

```
% Exact Green's function
G = sprintf('(%d*np.sin(%d))**(-1)*np.sin(%d*x)*np.sin(%d*(y-1))*(x<=y)+(%d*np.sin(
↪%d))**(-1)*np.sin(%d*y)*np.sin(%d*(x-1))*(x>y)',K,K,K,K,K,K,K,K);

% Output
output_example = {N, "ExactGreen", G};
```

### System of differential operators

The syntax is similar to define a system of differential operators. In this example, we define the following system:

$$\mathcal{L}(u,v) = \begin{pmatrix} \frac{d^2u}{dx^2} - v \\ -\frac{d^2v}{dx^2} + xu \end{pmatrix},$$

on the domain $[-1, 1]$ with boundary conditions:

$$u(-1) = 1,\ u(1) = -1,\ v(-1) = v(1) = -2.$$

```
function output_example = ODE_system()
% System of ODEs

% Define the domain.
dom = [-1, 1];

% Differential operator
N = chebop(@(x,u,v) [diff(u,2)-v; -diff(v,2)+x.*u], dom);

% Boundary conditions
N.bc = @(x,u, v) [u(-1)-1; u(1)+1; v(-1)+2; v(1)+2];

% Output
output_example = {N};
end
```

### Generating the dataset

Make sure that the MATLAB codes generate_gl_example.m and generate_gl_datasets.m are present in the parent directory of `examples/` and run the following command in a MATLAB terminal:

```
generate_gl_example("helmholtz");
```

Alternatively, all the datasets corresponding to the examples in the folder `examples/` can be generated as follows:

```
generate_gl_datasets();
```

The datasets are saved as `.mat` files at the location `examples/datasets/`.

- By default, the code generates 100 sampled functions $f$ from a squared-exponential Gaussian process, and solve the equation $\mathcal{L}u = f$ for the different forcing terms to obtain the training solutions $u$. Then, the forcing terms are evaluated at 200 uniform points in the domain $\Omega$, while the solutions are evaluated at 100 points.
- It is possible to edit the MATLAB script generate_gl_example.m to add noise to the output functions, or change the different parameters such as number of forcing terms, spatial points, . . .

### 2.2.2 Example 1: Helmholtz operator

After generating the training dataset `examples/datasets/helmholtz.m` corresponding to the helmholtz operator, we now try to discover its Green's function $G$ and associated homogeneous solution $u_{\text{hom}}$ such that

$$\mathcal{L}u = f \Longleftrightarrow u(x) = \int_{\Omega} G(x,y)f(y)\mathrm{d}y + u_{\text{hom}}(x), \quad \forall x \in \Omega,$$

where $u_{\text{hom}}$ is the solution to the homogeneous equation $\mathcal{L}u = 0$ satisfying the boundary conditions.

In this example, the prescribed boundary conditions are homogeneous Dirichlet on the interval $\Omega = [0, 1]$ and therefore $u_{\text{hom}} = 0$.

#### Training the neural networks

The neural networks can be implemented and trained by running the following Python script:

```python
# Import the library
import greenlearning as gl

# Construct neural networks for G and homogeneous solution
G_network = gl.matrix_networks([2] + [50] * 4 + [1], "rational", (1,1))
U_hom_network = gl.matrix_networks([1] + [50] * 4 + [1], "rational", (1,))

# Define the model
model = gl.Model(G_network, U_hom_network)

# Train the model on the dataset "helmholtz" in the path "examples/datasets/"
model.train("examples/datasets/","helmholtz")

# Save the training loss
model.save_loss()

# Plot the results
model.plot_results()

# Save the NNs evaluated at a grid in a csv file
model.save_results()

# Close the TensorFlow session
model.sess.close()
```

We then create one neural network for the Green's function: `G_network`, which accepts two inputs $(x, y)$ and returns one output: $G(x, y)$. This network has 4 hidden layers of 50 neurons each and uses rational activation functions (see the paper about rational neural networks for more details).

The last parameter of `matrix_networks` is the shape `(n_output, n_input)` corresponding to the dimension of the problem, where `n_input` is the number of input forcing terms $f$ to the system and `n_output` is the number of solutions $u$. In this example, there is only one input and one output (as it is a scalar ODE) so we set the shape to `(1,1)`.

The shape of the homogeneous solution network must be equal to the number of output functions $u$: `(n_output,)`.

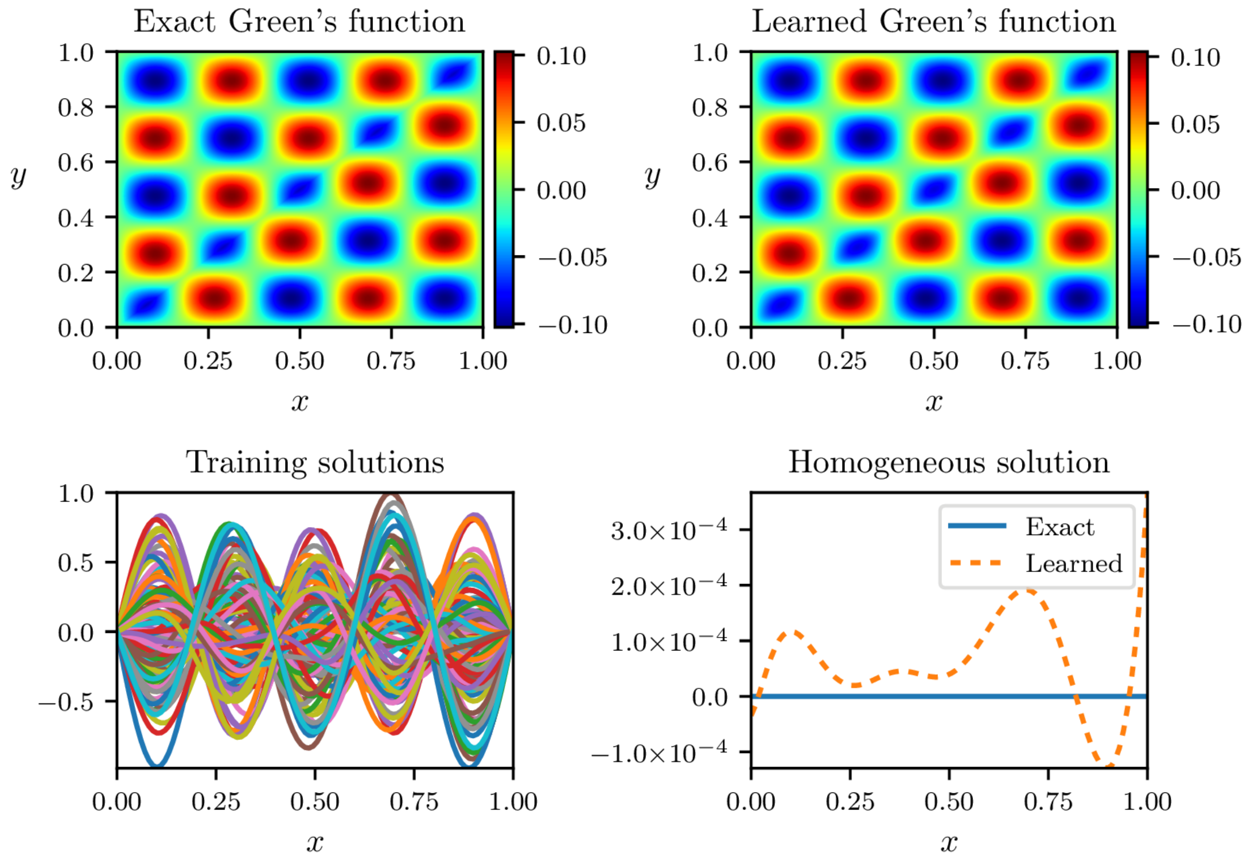
By default, the networks are trained using first 1000 steps of Adam's optimizer and then up to $5 \times 10^4$ steps of L-BFGS-B optimizer.

### Visualizing the Green's function

The line

```
model.plot_results()
```

displays the learned Green's function (top-right panel), the training solutions $u$ used (bottom-left), and the learned homogeneous solution (bottom-right) in the file `results/helmholtz_rational.pdf`.



*Exact and learned Green's function of the Helmholtz operator using a rational neural network.*

- Using rational neural networks for both the Green's function and homogeneous solution, we achieve a relative error between the exact and learned Green's function of $0.9\%$. The relative error is defined as

$$\text{Relative error} = \frac{||G_{\text{exact}} - G_{\text{learned}}||_{L^2(\Omega)}}{||G_{\text{exact}}||_{L^2(\Omega)}}$$

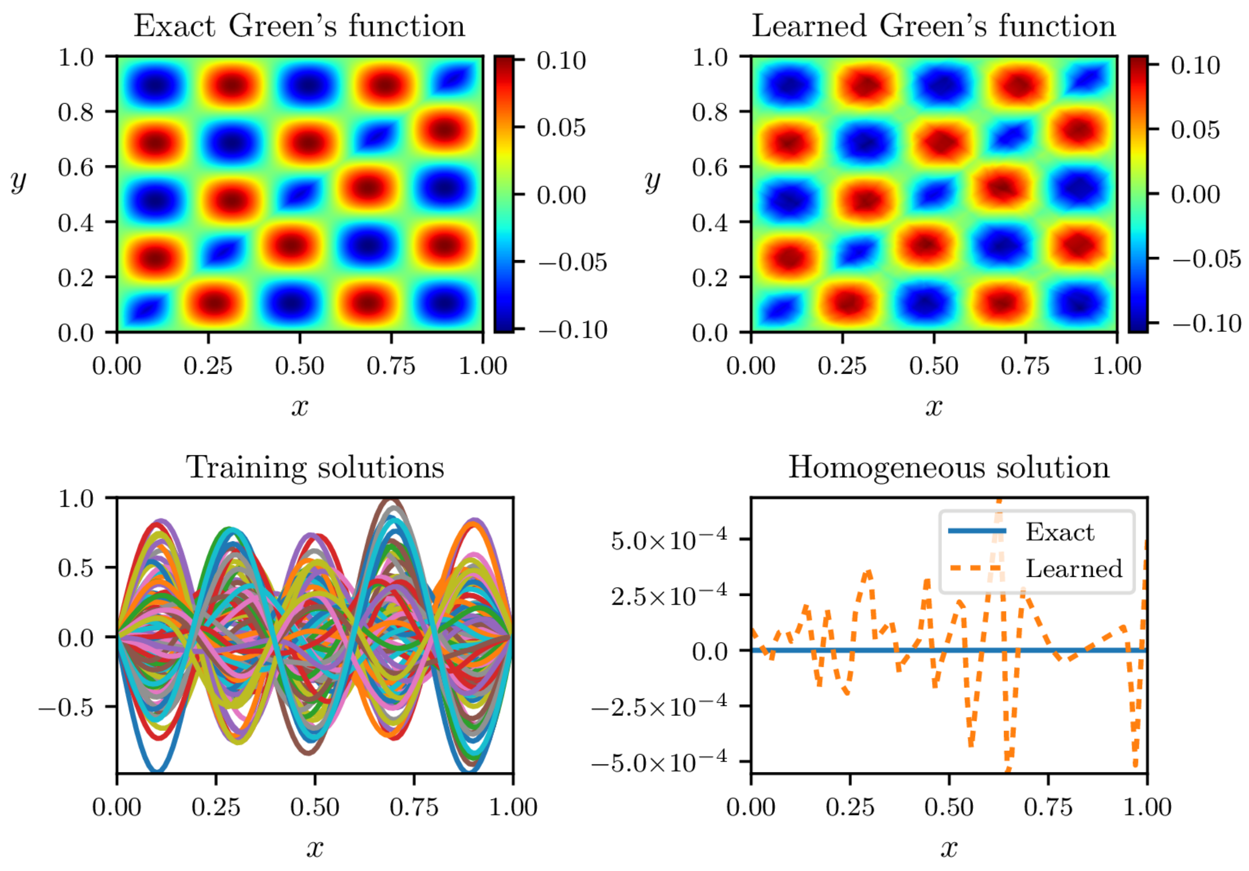
- The values of the loss function during the training process are saved in the following file: `training/loss_rational.csv`.

### ReLu vs Rational neural network

We can also specify a different activation function, (e.g. ReLU) for the neural networks easily as

```
# Construct neural networks for G and homogeneous solution
G_network = gl.matrix_networks([2] + [50] * 4 + [1], "relu", (1,1))
U_hom_network = gl.matrix_networks([1] + [50] * 4 + [1], "relu", (1,))
```

and we obtain the following figure.



*Exact and learned Green's function of the Helmholtz operator using a ReLU neural network.*

Here, the relative error reaches $5.3\%$, which is significantly larger than the one obtained with rational neural networks. In addition, the Green's function and homogeneous solution learned with ReLU networks are not necessarily smooth as shown by the figure above.

### 2.2.3 Example 2: System of ODEs

In this example, we learn a matrix $G$ of Green's functions and homogeneous solutions $(u_{\text{hom}}, v_{\text{hom}})$ associated to the system of differential operators:

$$\mathcal{L}(u,v) = \begin{pmatrix} \frac{d^2u}{dx^2} - v \\ -\frac{d^2v}{dx^2} + xu \end{pmatrix},$$

on the domain $[-1,1]$ with boundary conditions:

$$u(-1) = 1,\ u(1) = -1,\ v(-1) = v(1) = -2.$$

Hence, if $f_1$ and $f_2$ are two forcing terms, then the solution $(u, v)$ to the equations

$$\frac{d^2u}{dx^2} - v = f_1, \tag{2.1}$$

$$-\frac{d^2v}{dx^2} + xu = f_2 \tag{2.2}$$

satisfies

$$\begin{pmatrix} u(x) \\ v(x) \end{pmatrix} = \int_{[-1,1]} \begin{pmatrix} G_{0,0}(x,y) & G_{0,1}(x,y) \\ G_{1,0}(x,y) & G_{1,1}(x,y) \end{pmatrix} \begin{pmatrix} f_1(y) \\ f_2(y) \end{pmatrix} \mathrm{d}y + \begin{pmatrix} u_{\text{hom}}(x) \\ v_{\text{hom}}(x) \end{pmatrix},$$

for all $x \in [-1, 1]$. Note that we have employed the Python indexing notation for matrices. Moreover, the homogeneous solutions satisfy the following equations:

### Implementation

We implement this example in `GreenLearning` by running the following Python script:

```python
# Import the library
import greenlearning as gl

# Construct neural networks for G and homogeneous solution
G_network = gl.matrix_networks([2] + [50] * 4 + [1], "rational", (2,2))
U_hom_network = gl.matrix_networks([1] + [50] * 4 + [1], "rational", (2,))

# Define the model
model = gl.Model(G_network, U_hom_network)

# Train the model on the dataset "ODE_system" in the path "examples/datasets/"
model.train("examples/datasets/","ODE_system")

# Plot the results
model.plot_results()

# Close the Tensorflow session
model.sess.close()
```

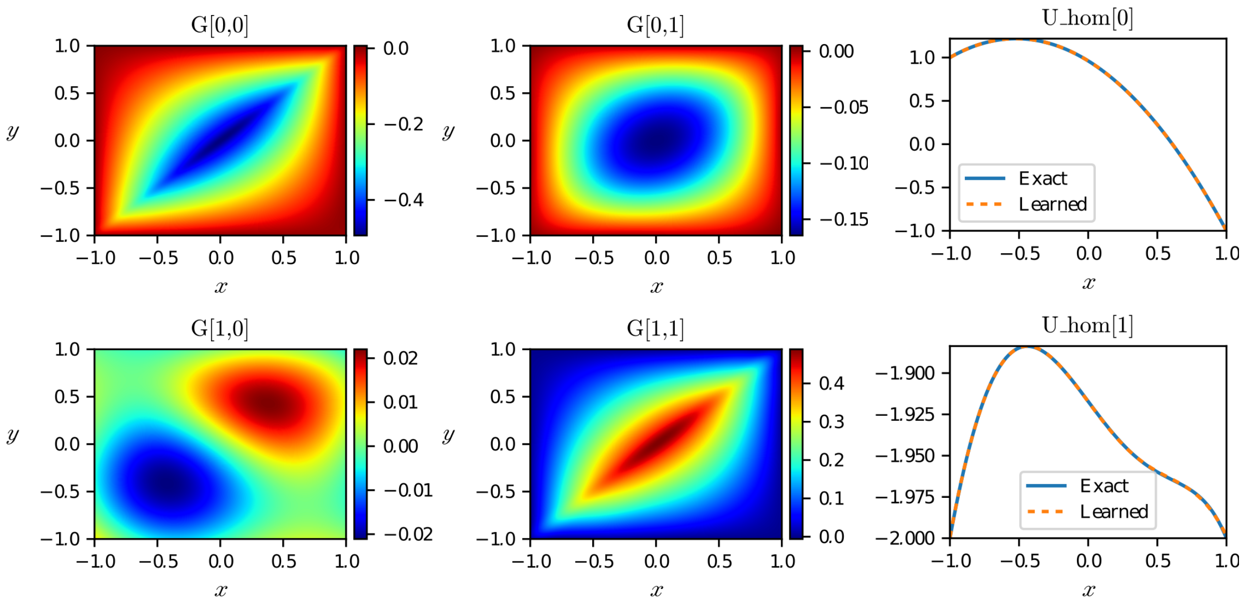
In particular, we defined the shape of the matrix of neural networks `G_network` to be `(2,2)` because there are two input functions $(f_1, f_2)$, and two outputs $(u, v)$. Similarly, the homogeneous solution network `U_hom_network` has shape `(2,)` to match the number of outputs.

### Numerical results

In the script written above, we used the following command to plot the Green's function and save the results:

```python
# Plot the results
model.plot_results()
```

Note that this command differs to the case of scalar operator (i.e. one input and output function) as it displays the matrix of Green's functions and the two homogeneous solutions.



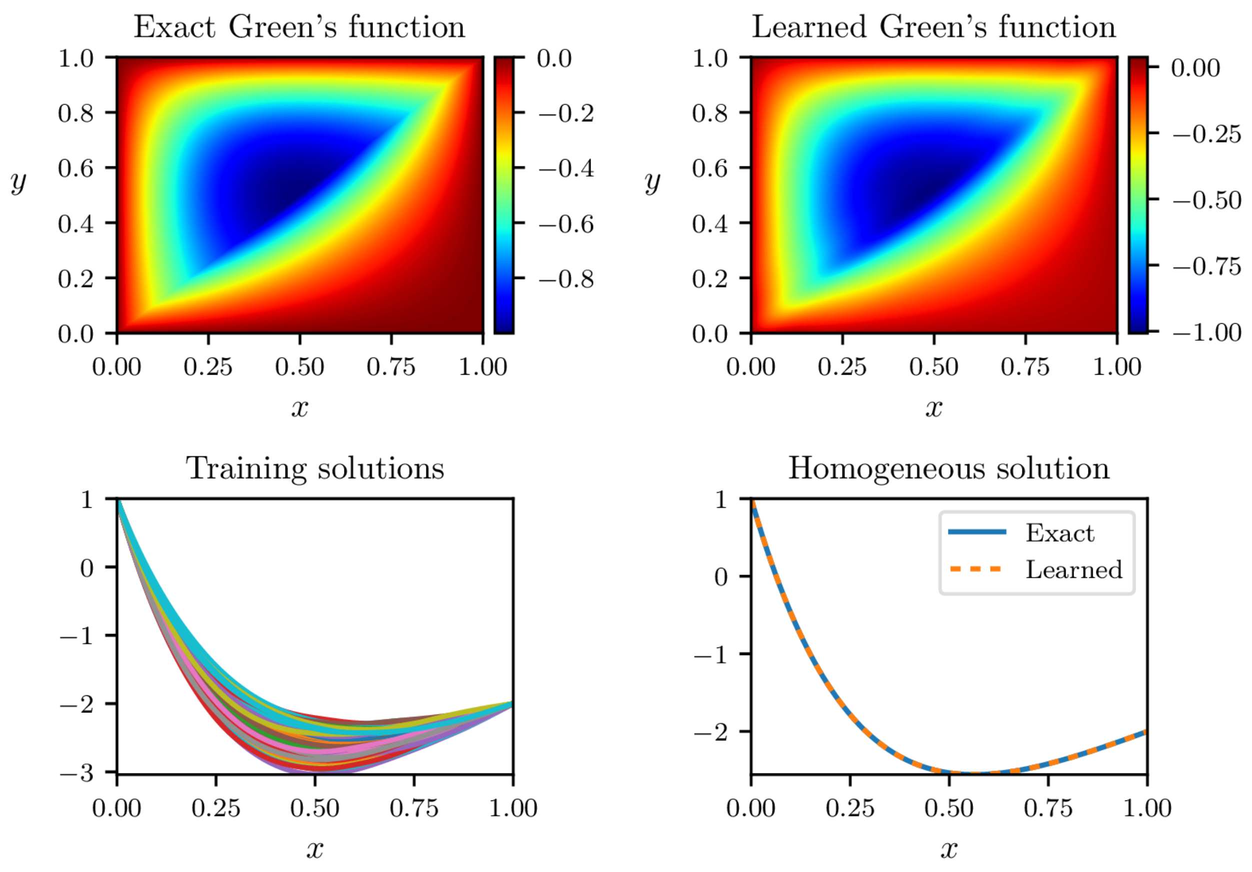
*Matrix of Green's functions together with homogeneous solutions learned by a rational neural network.*

In this figure, we recognize the diagonal blocks of the matrix of Green's functions, which correspond to the Laplacian and negative Laplacian operator. We also see on the right panel that the homogeneous solutions to the system of ODEs are learned by the rational neural networks with high accuracy.

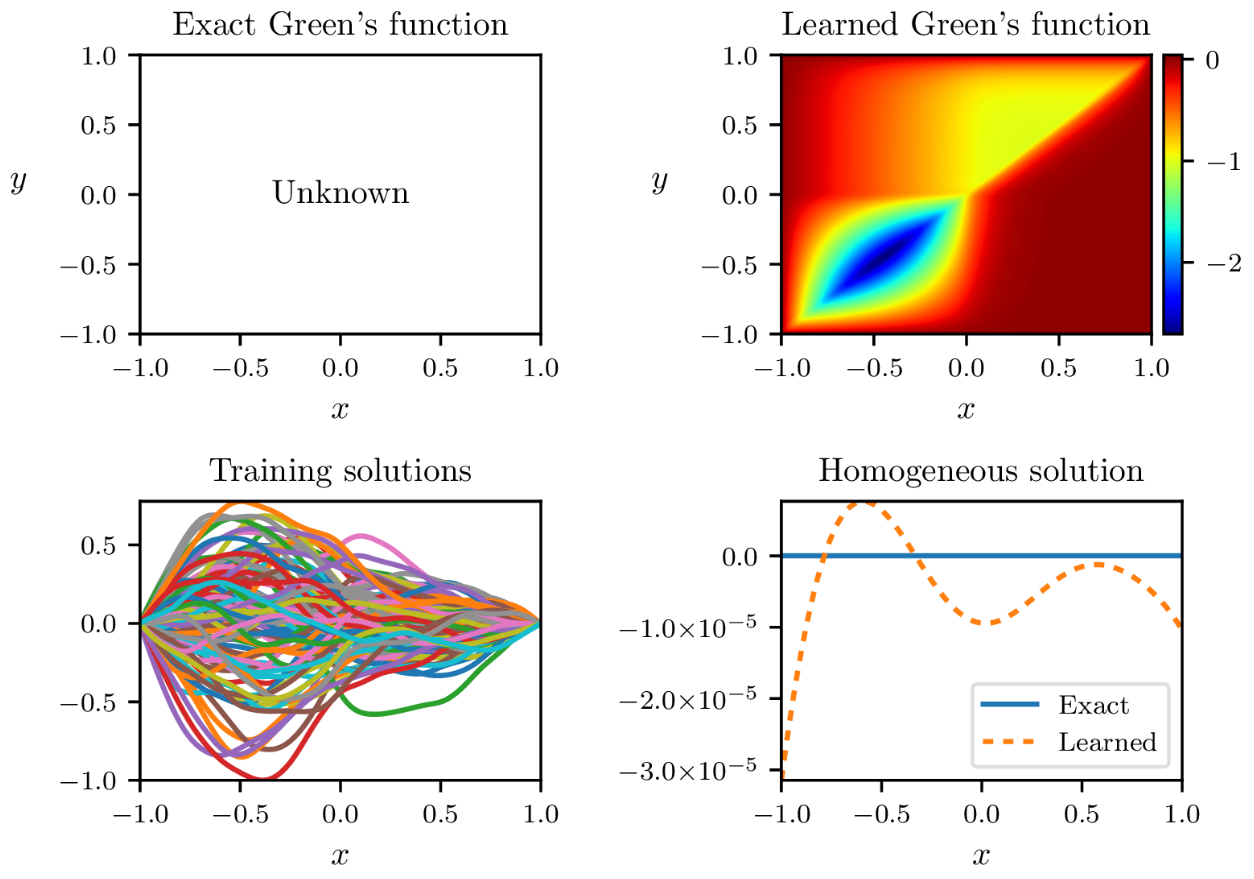
## 2.3 Gallery of examples

In this section, we display the Green's functions of the examples located in `examples/` learned by GreenLearning. Each title is a link to the corresponding MATLAB script.

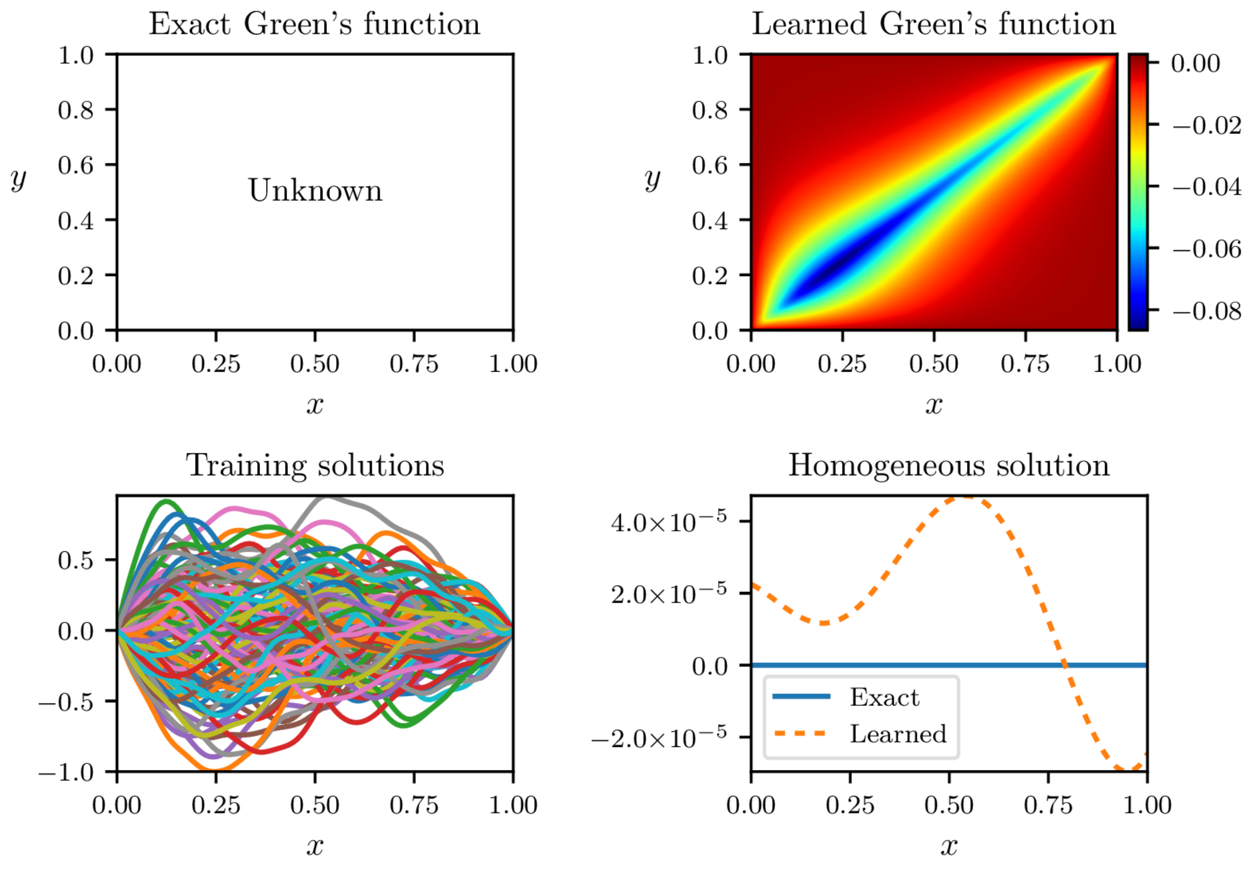
### 2.3.1 advection_diffusion



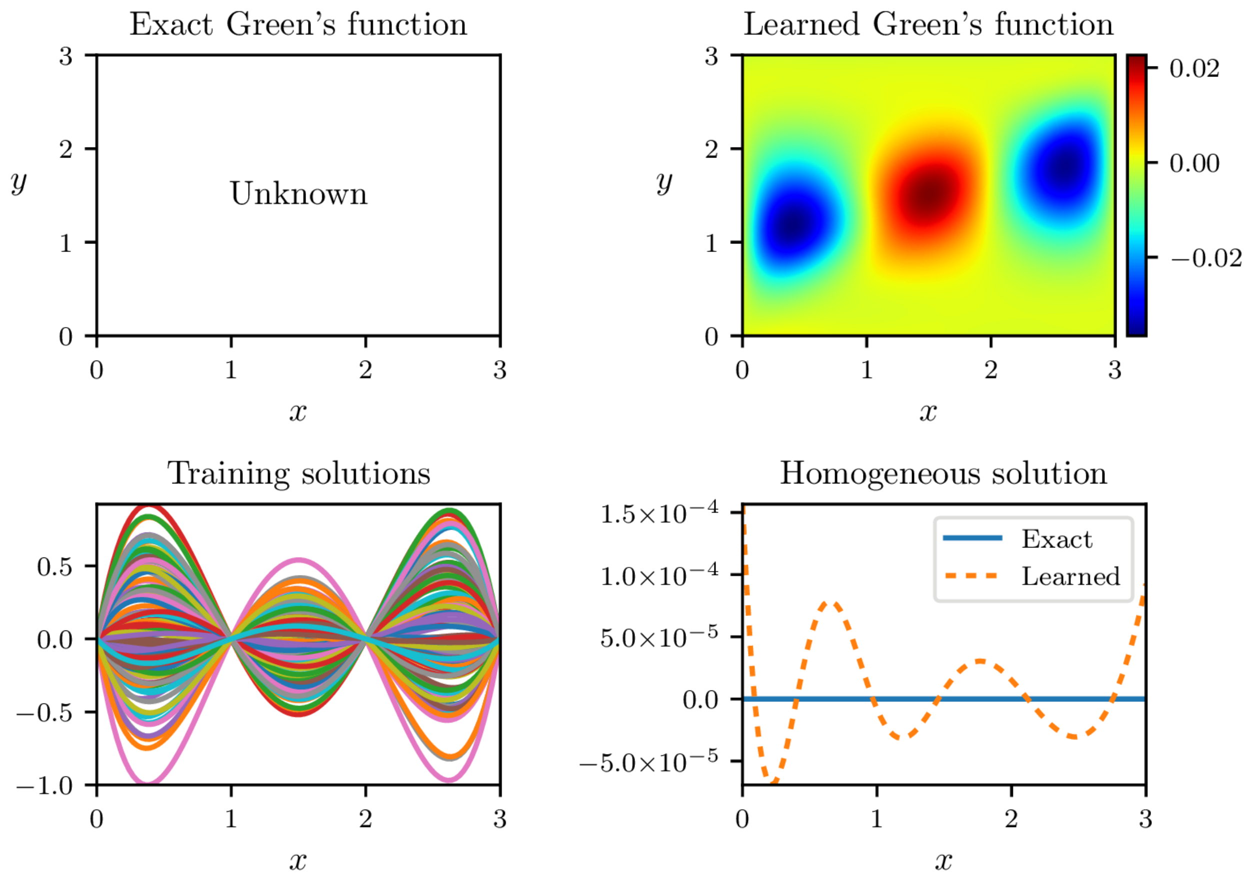
### 2.3.2 advection_diffusion_jump



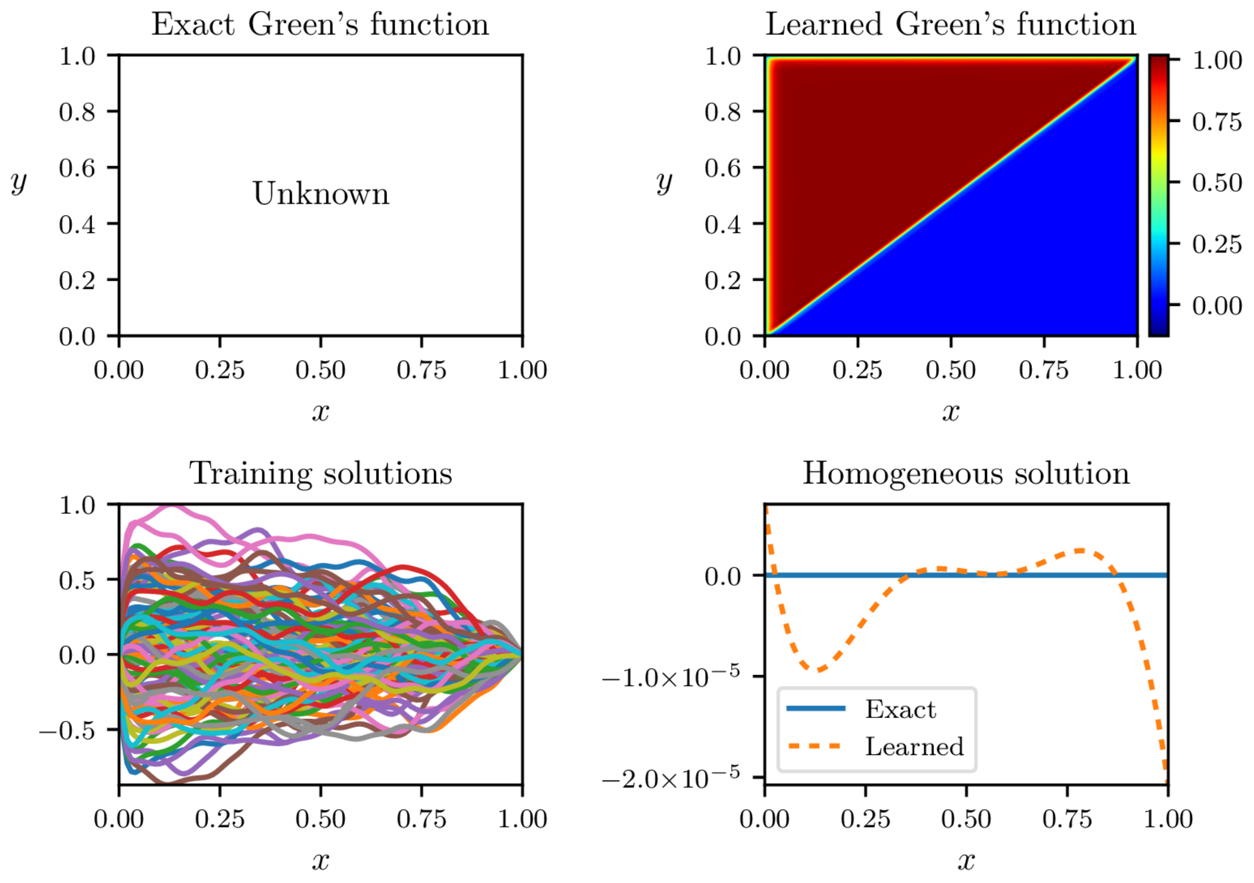
### 2.3.3 airy_equation



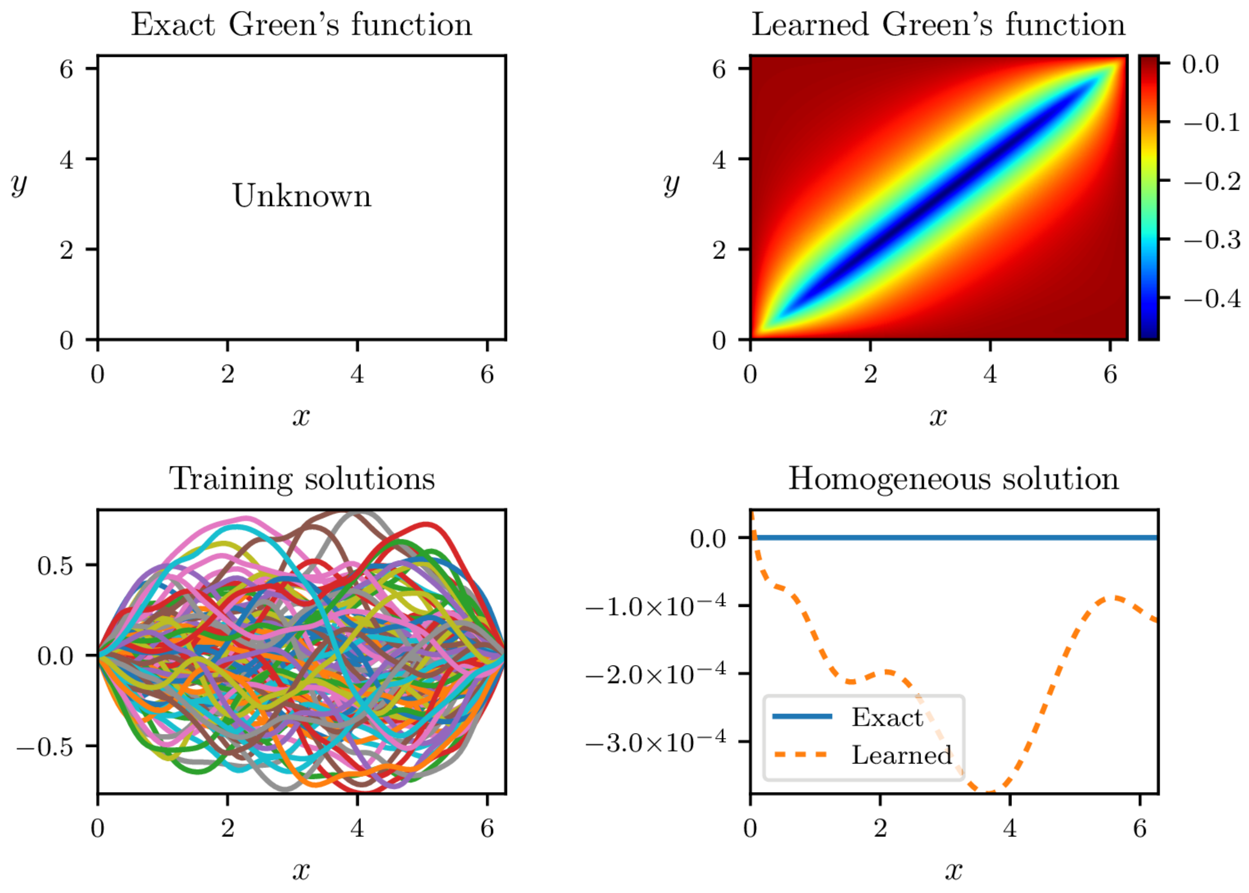
### 2.3.4 biharmonic



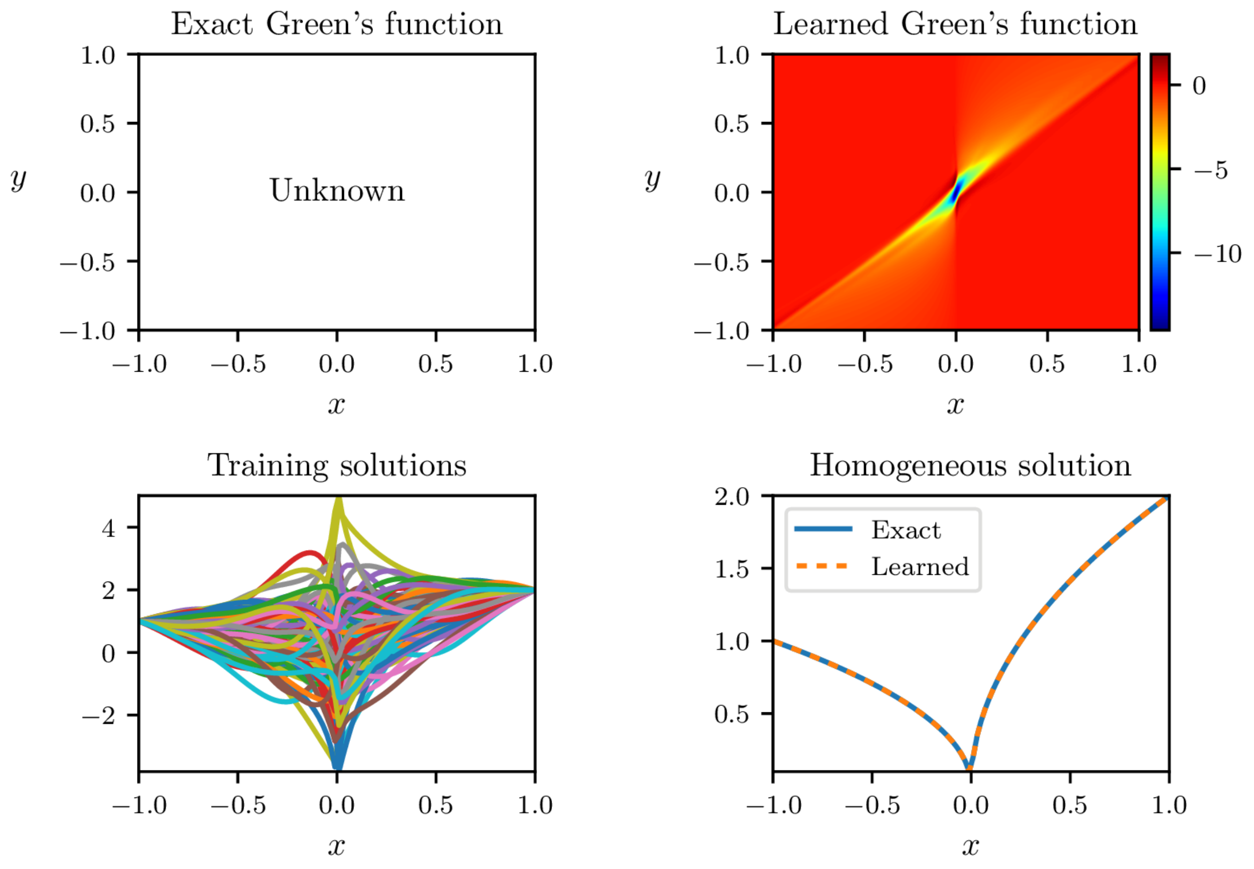
### 2.3.5 boundary_layer



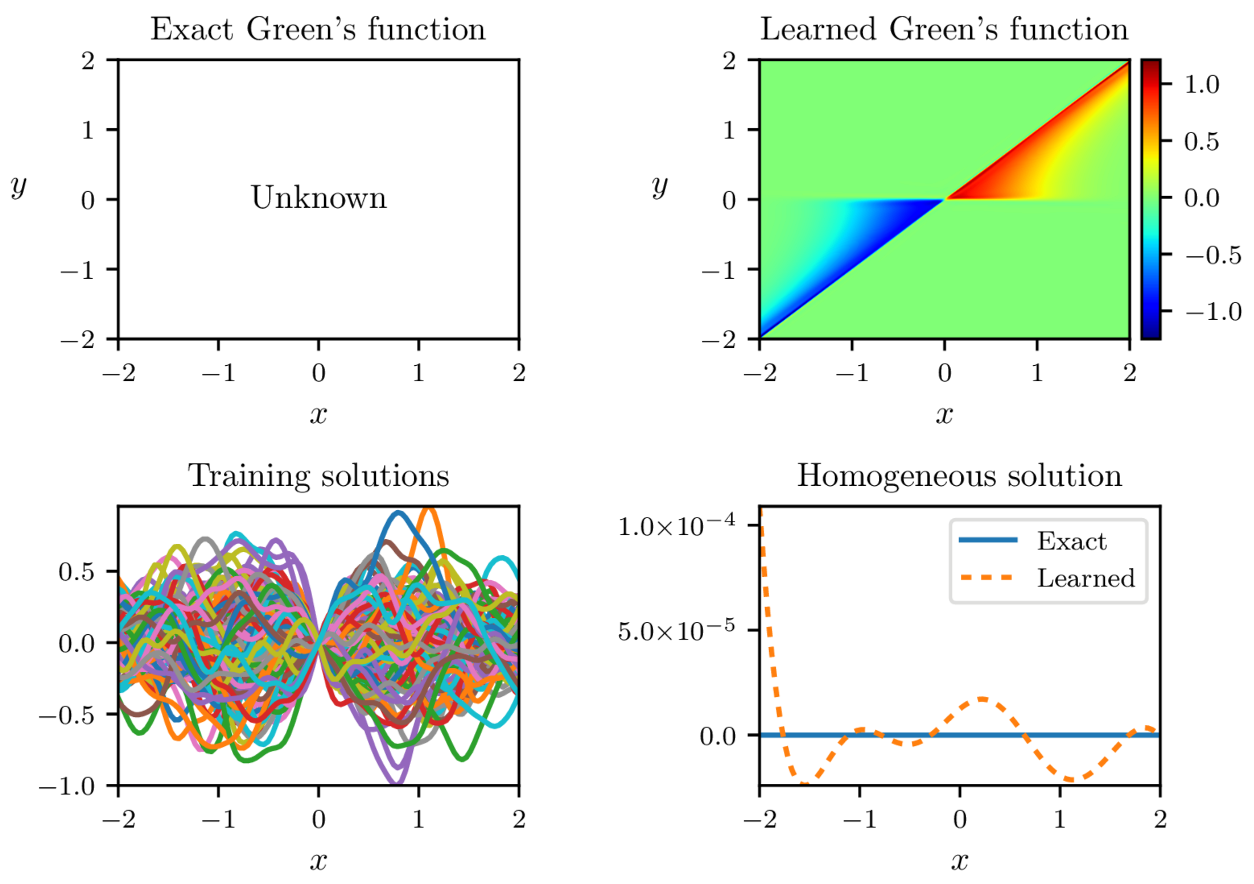
### 2.3.6 cubic_helmholtz



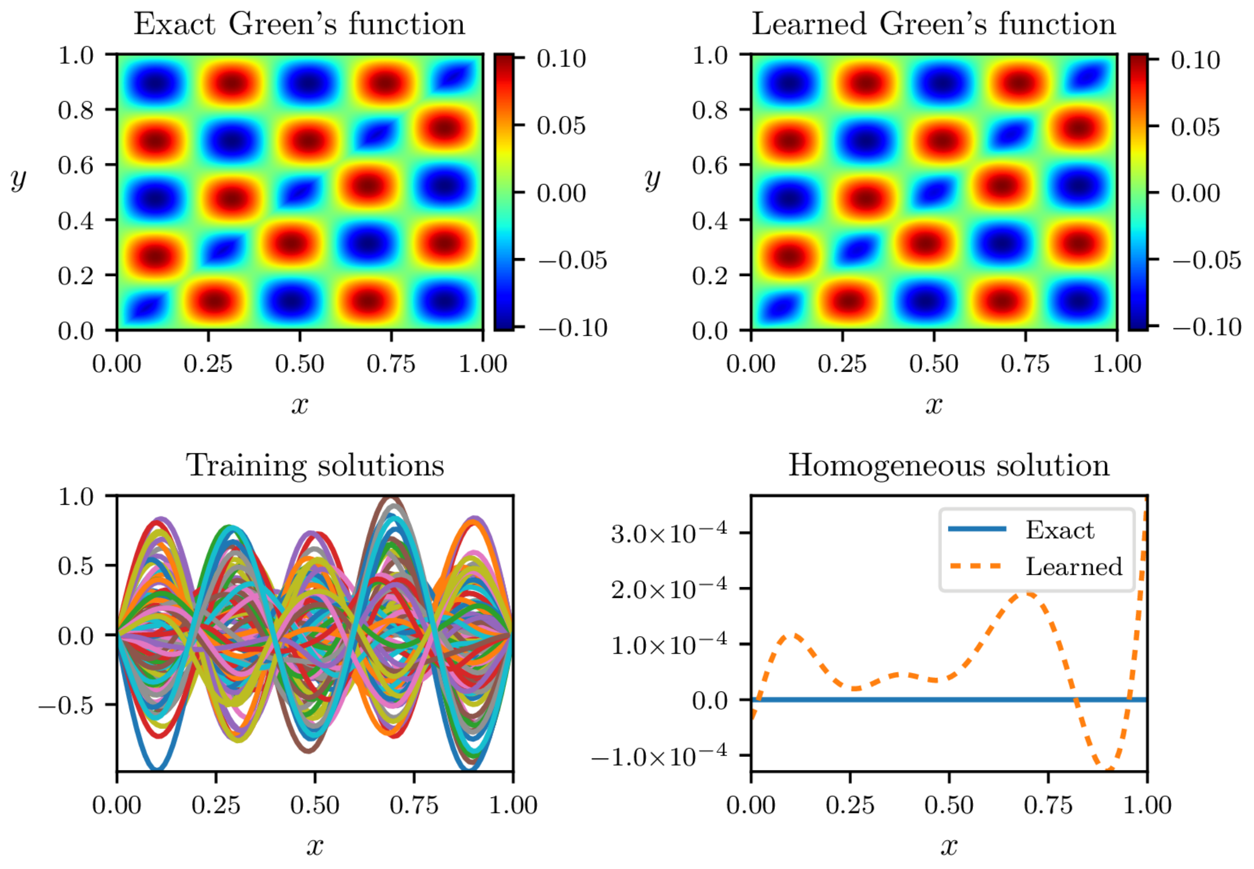
### 2.3.7 cusp


Exact Green's function
Unknown
Learned Green's function
Training solutions
Homogeneous solution
Exact
Learned
x
y

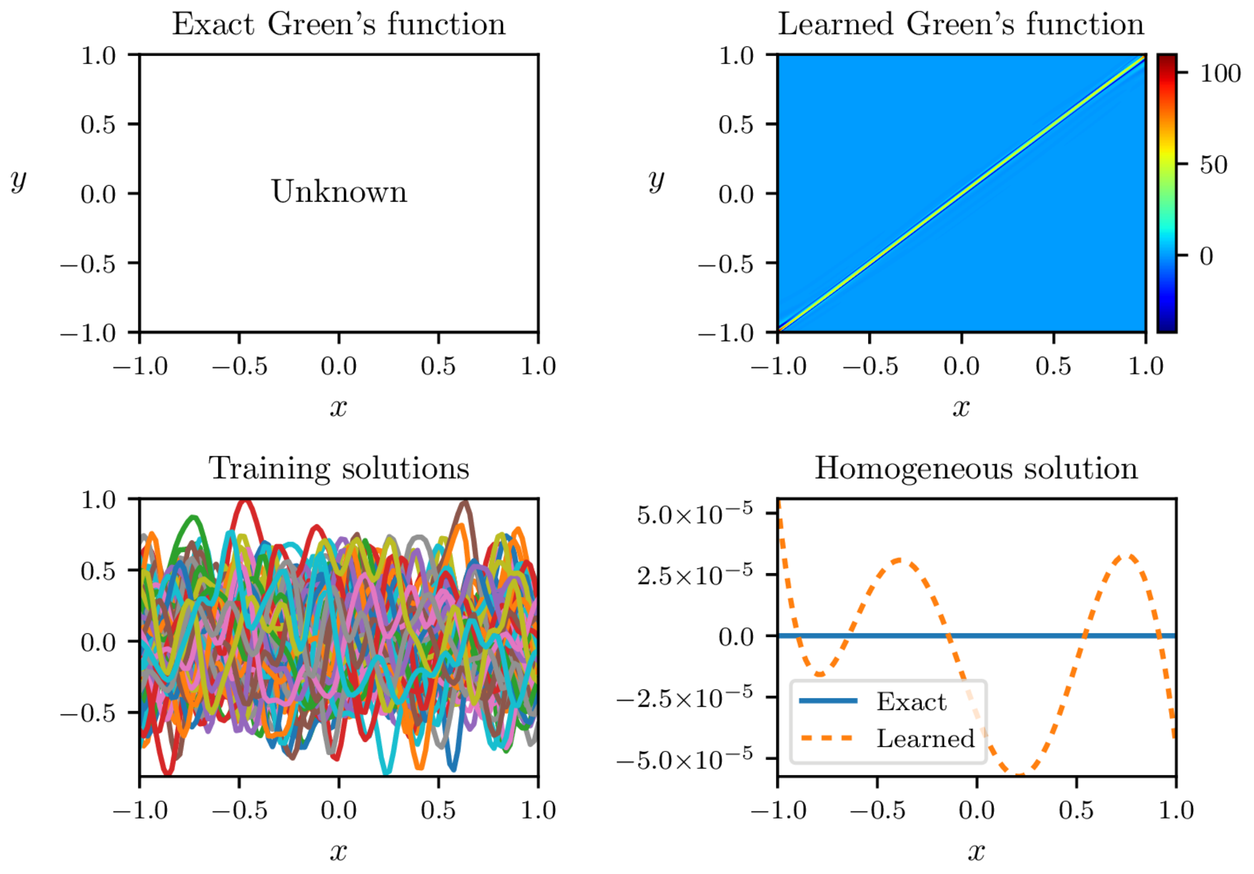
### 2.3.8 dawson


Exact Green's function
Unknown
Learned Green's function
Training solutions
Homogeneous solution
Exact
Learned
$x$
$y$

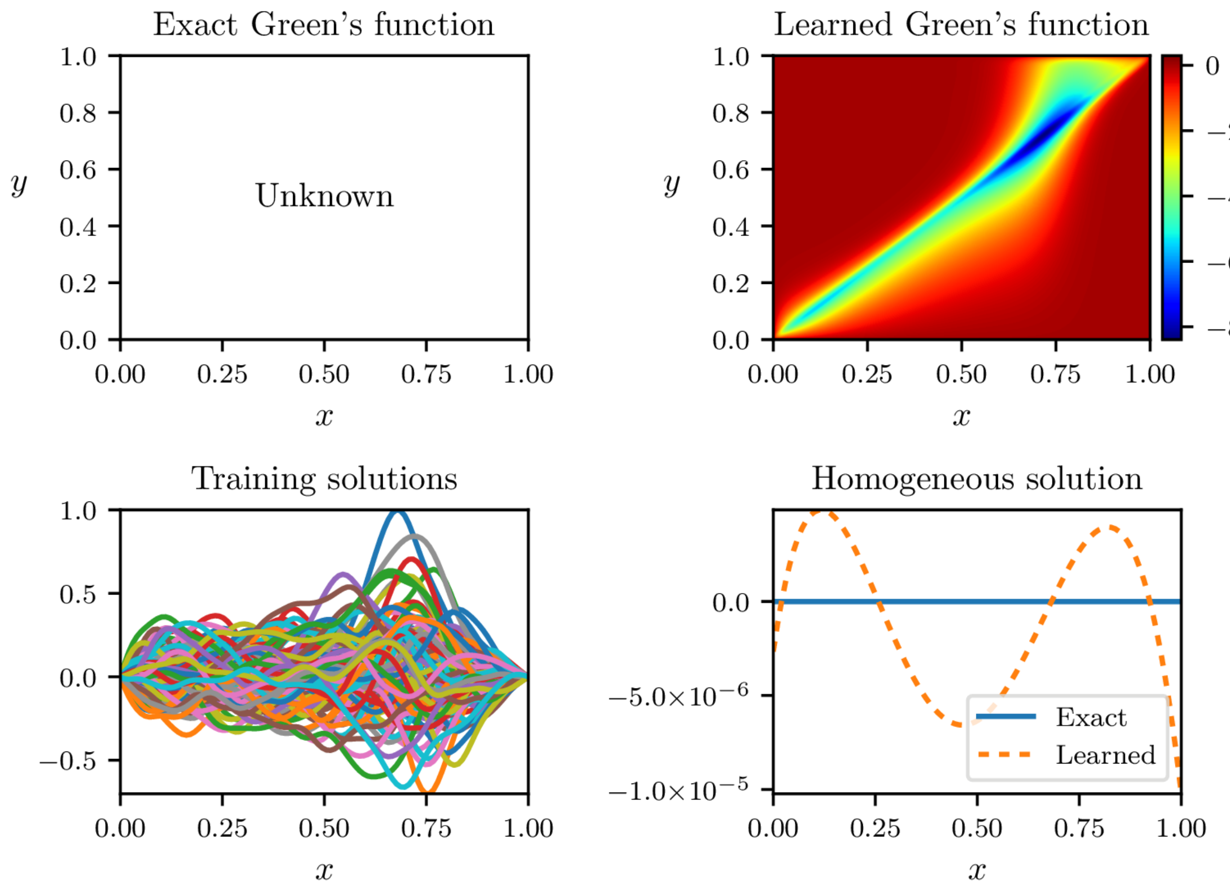
### 2.3.9 helmholtz



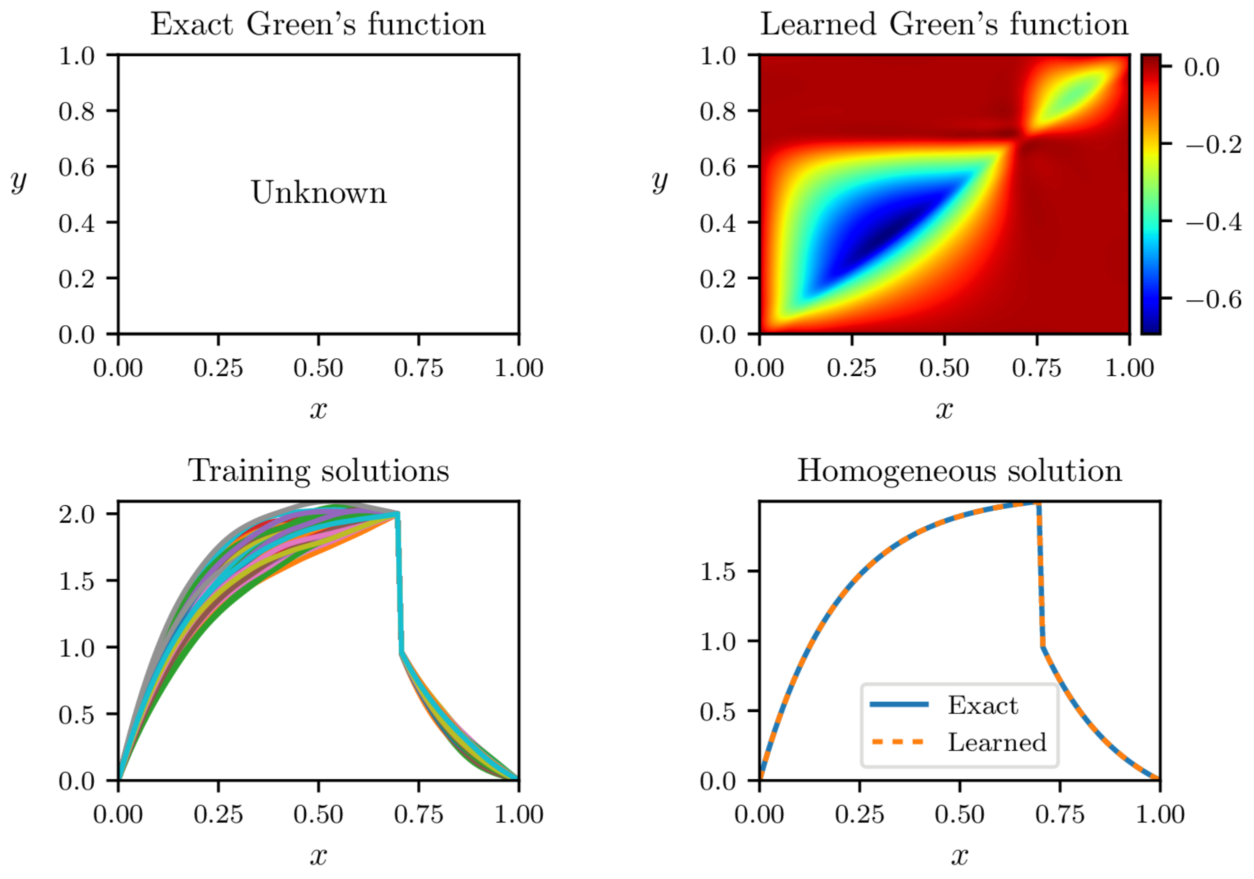
### 2.3.10 identity



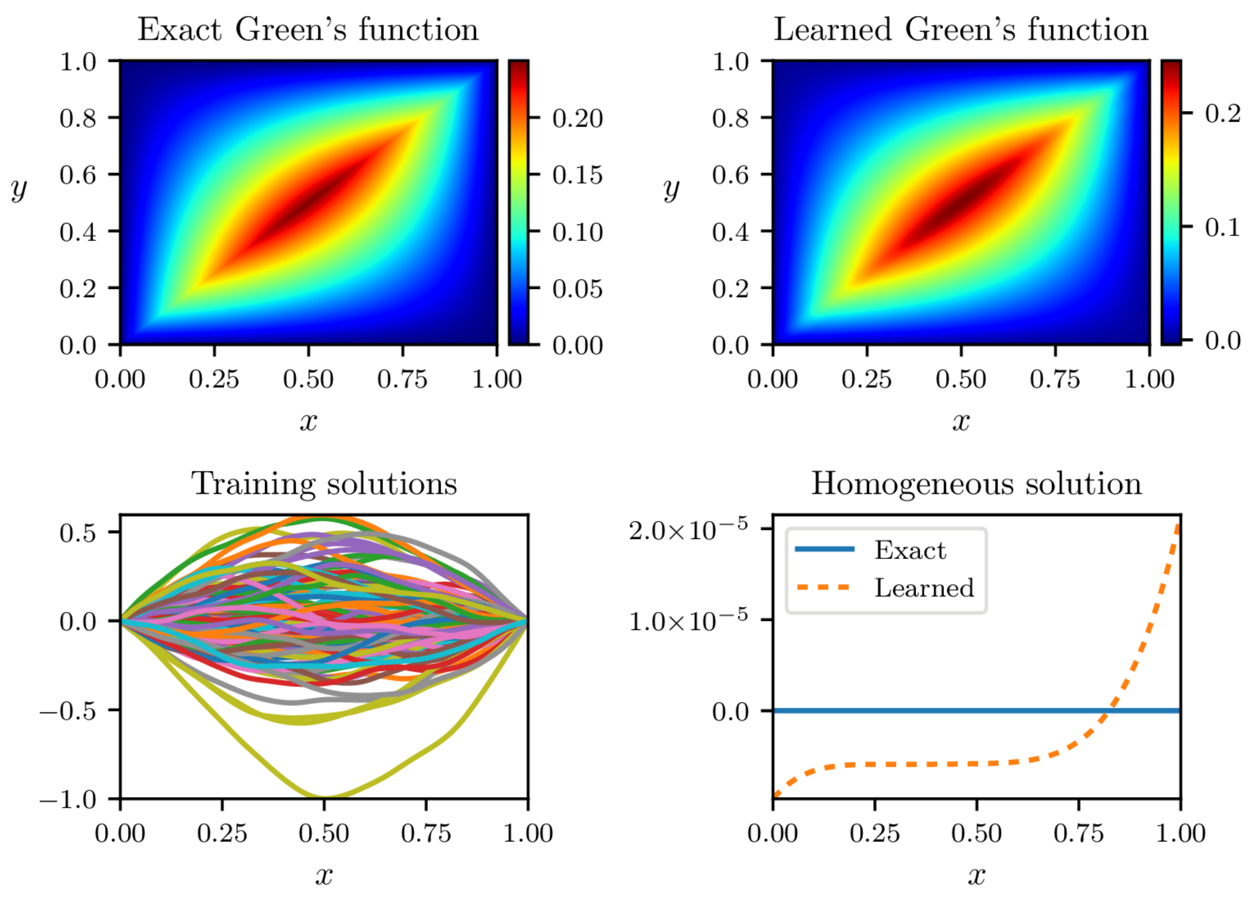
### 2.3.11 interior_layer



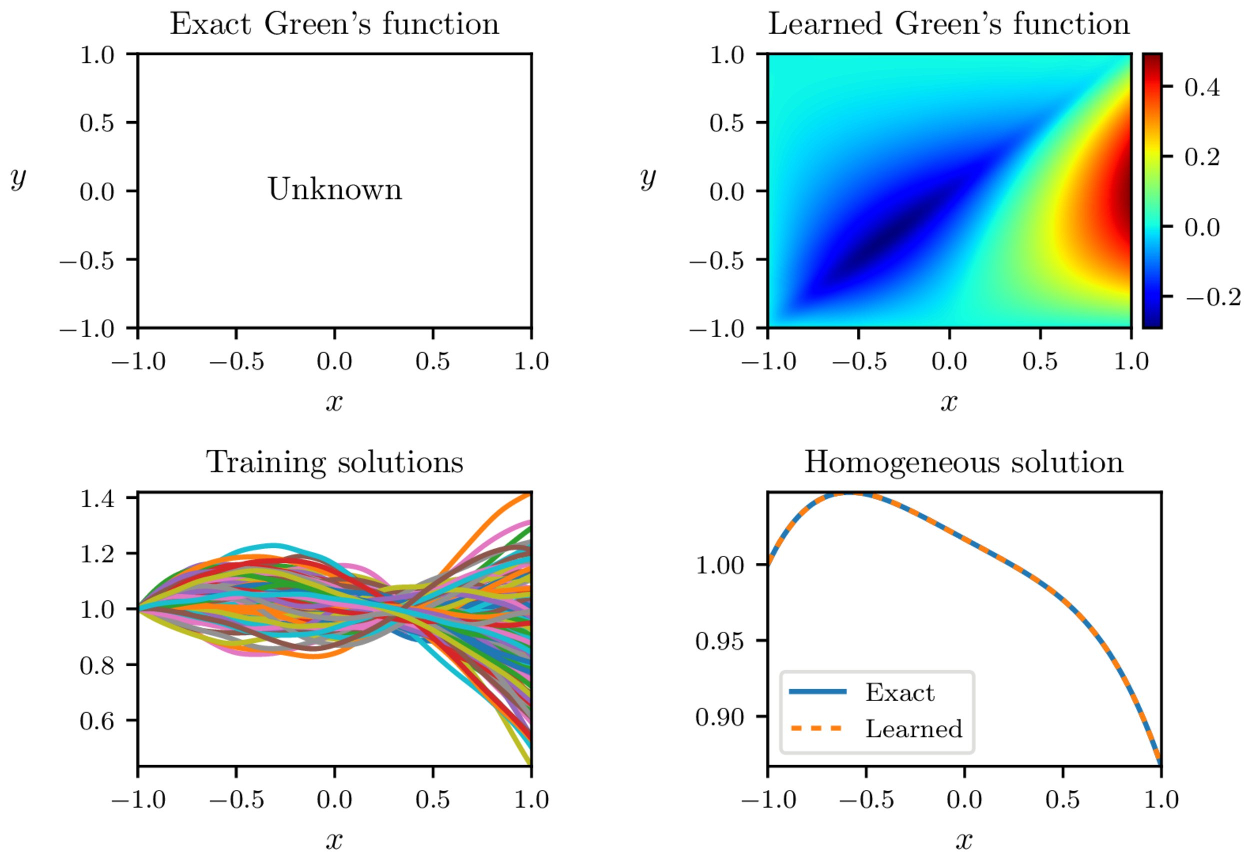
### 2.3.12 jump_green



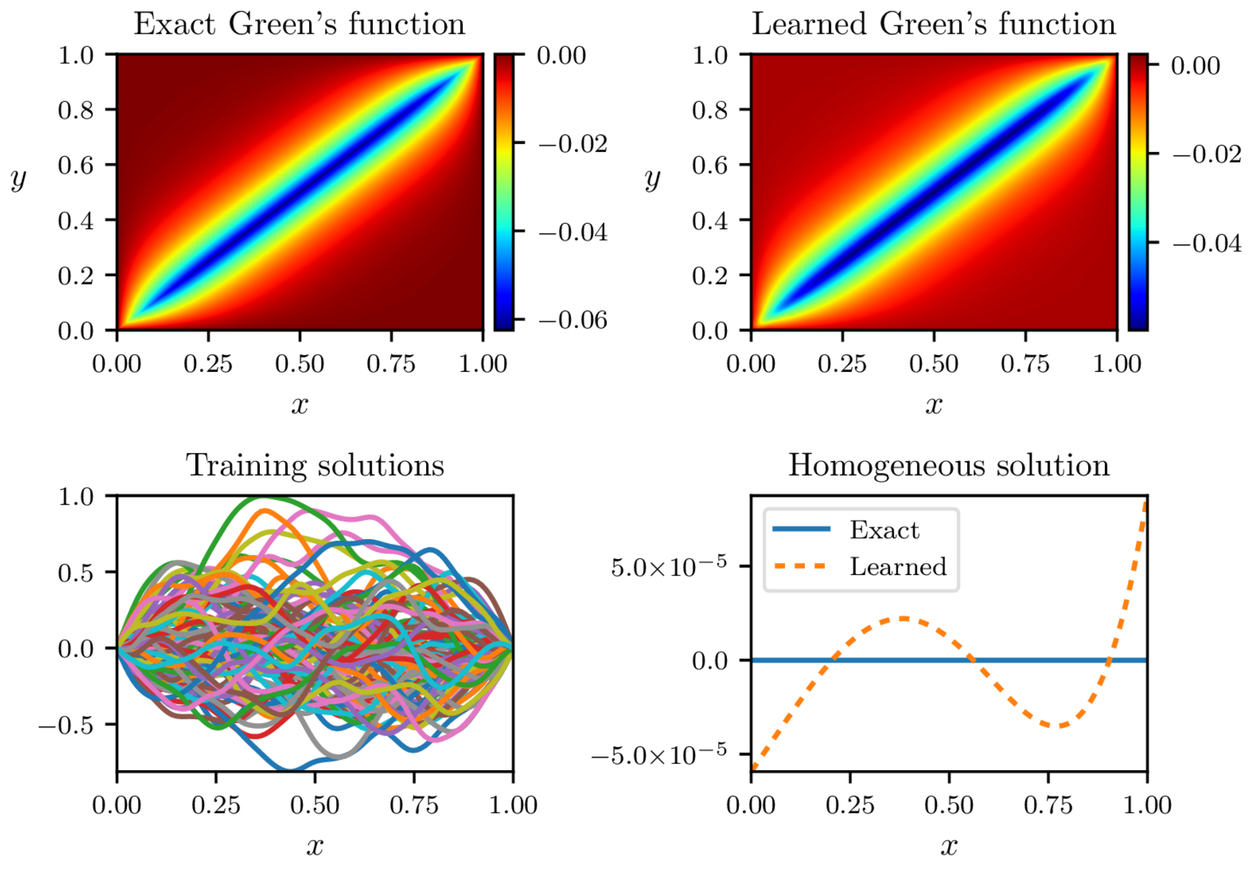
### 2.3.13 laplace



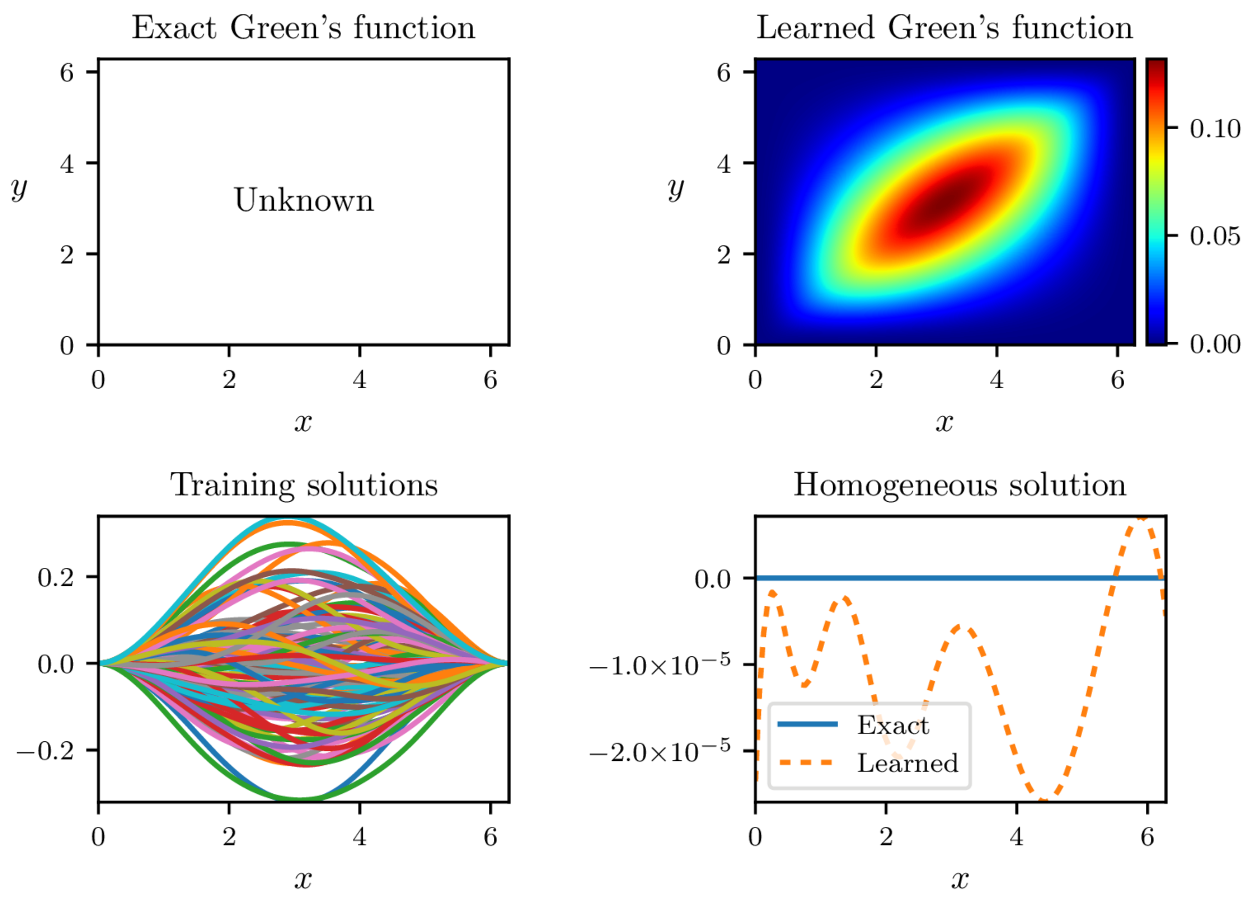
### 2.3.14 mean_condition



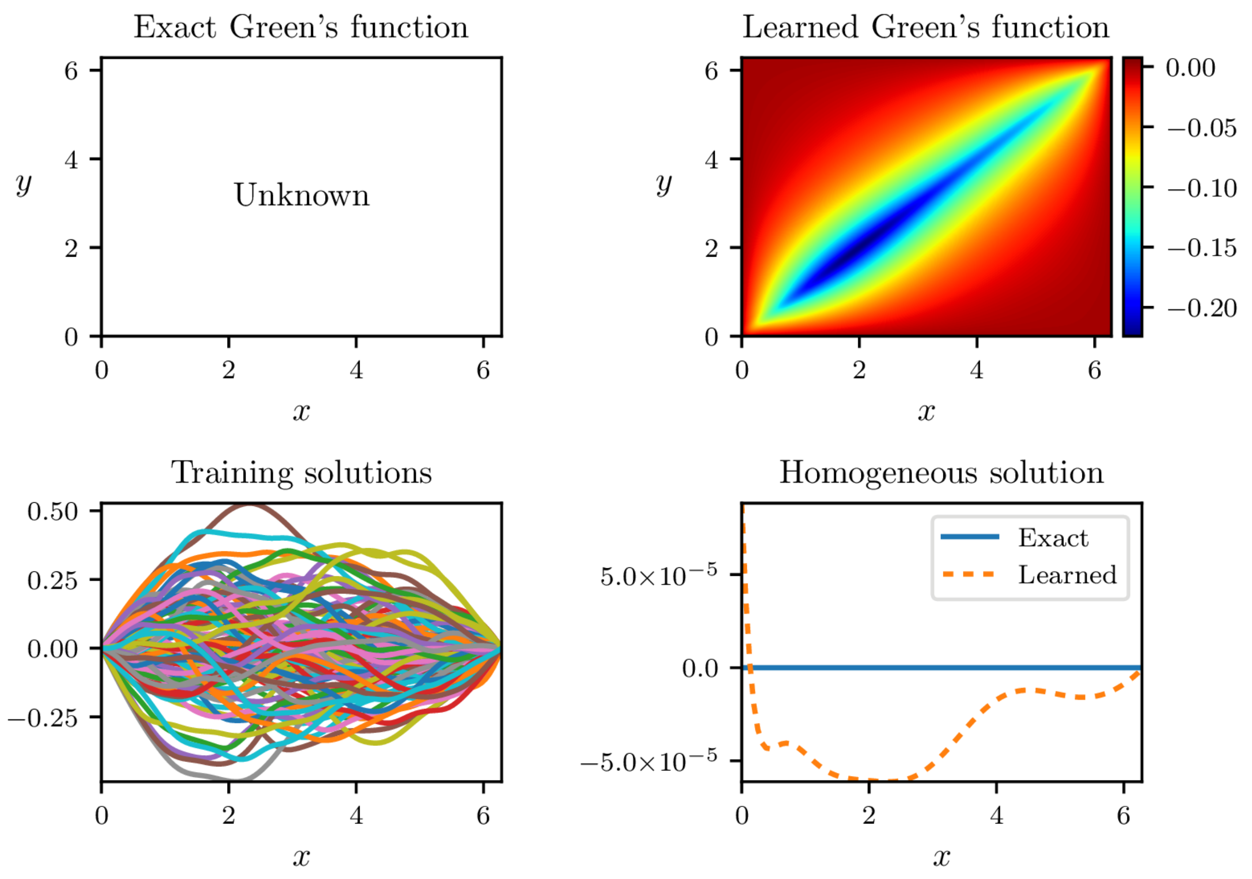
### 2.3.15 negative_helmholtz



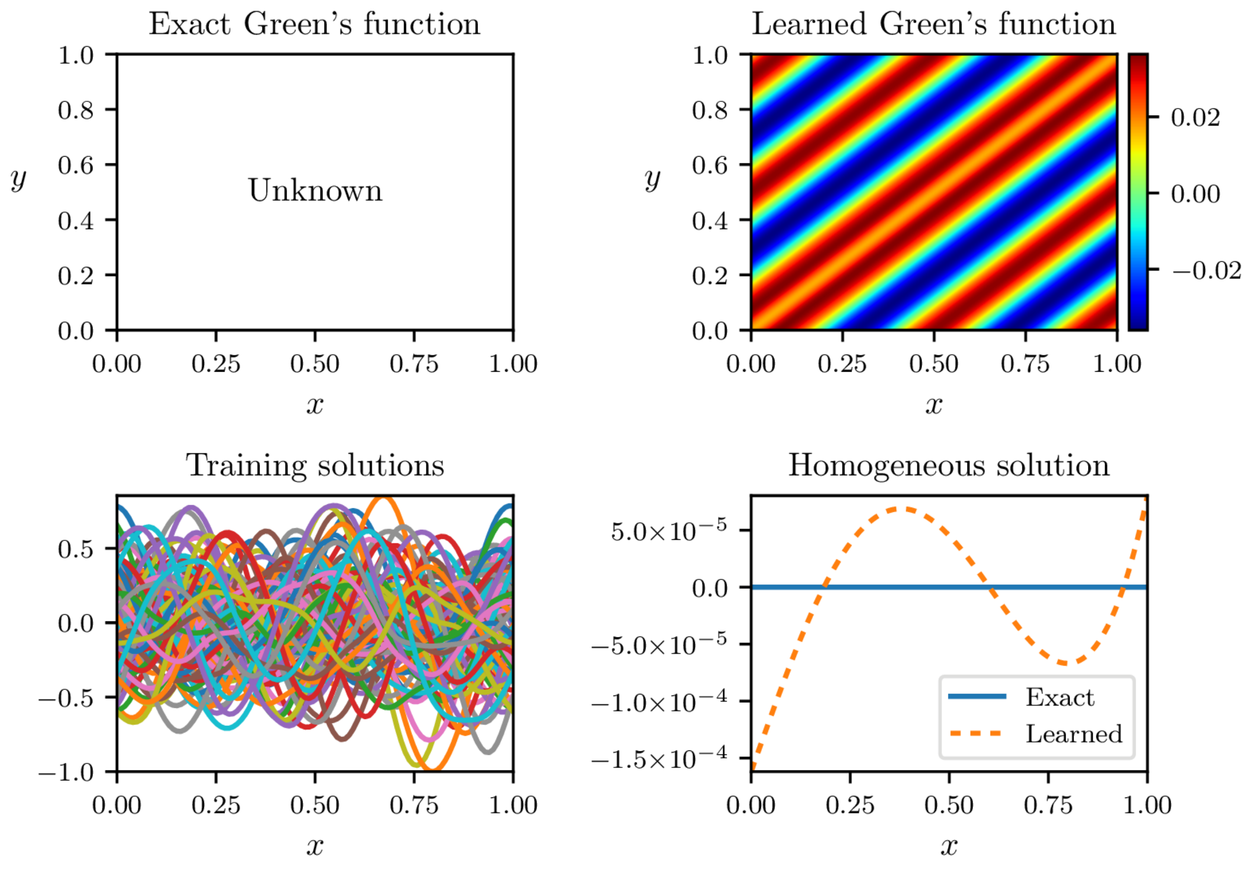
### 2.3.16 nonlinear_biharmonic



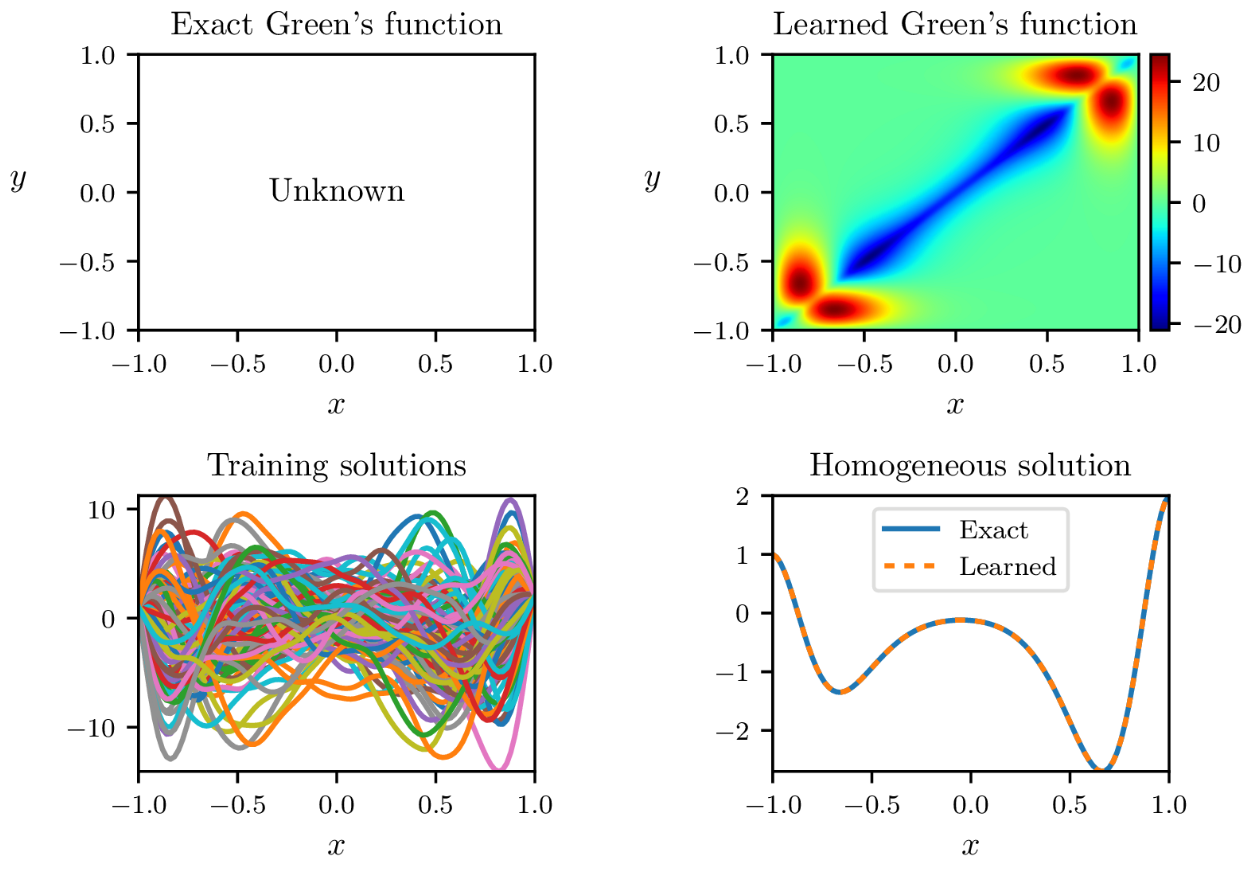
### 2.3.17 nonlinear_SL



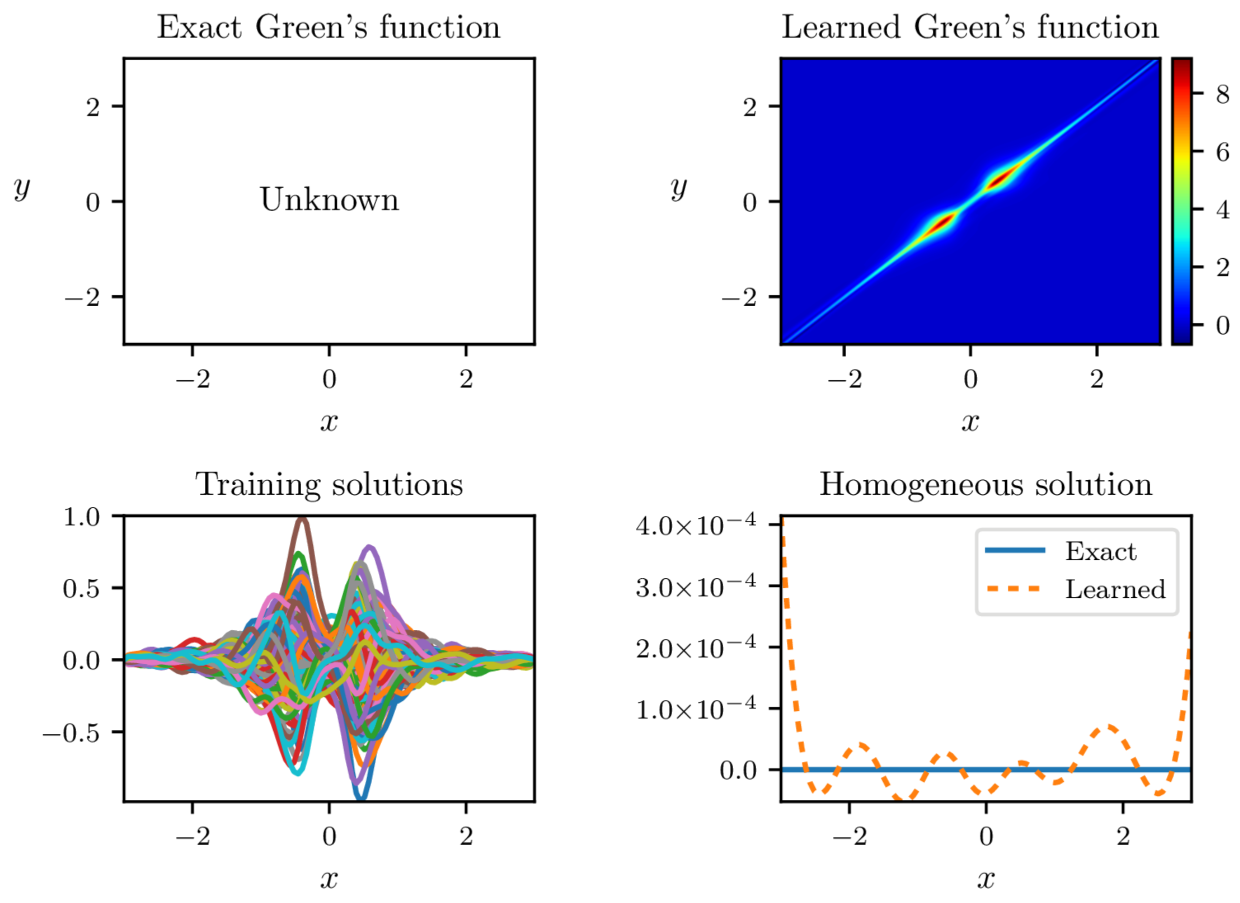
### 2.3.18 periodic_helmholtz



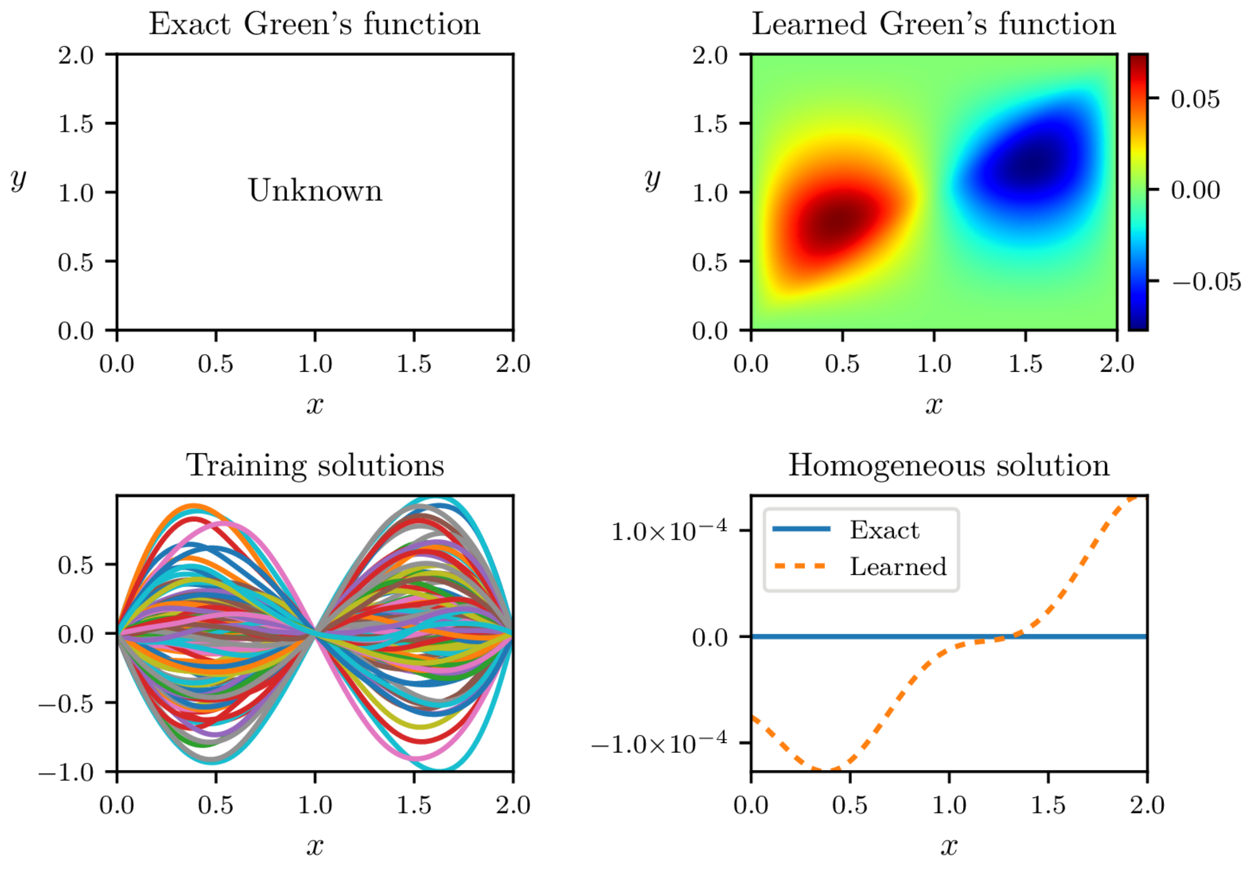
### 2.3.19 potential_barrier



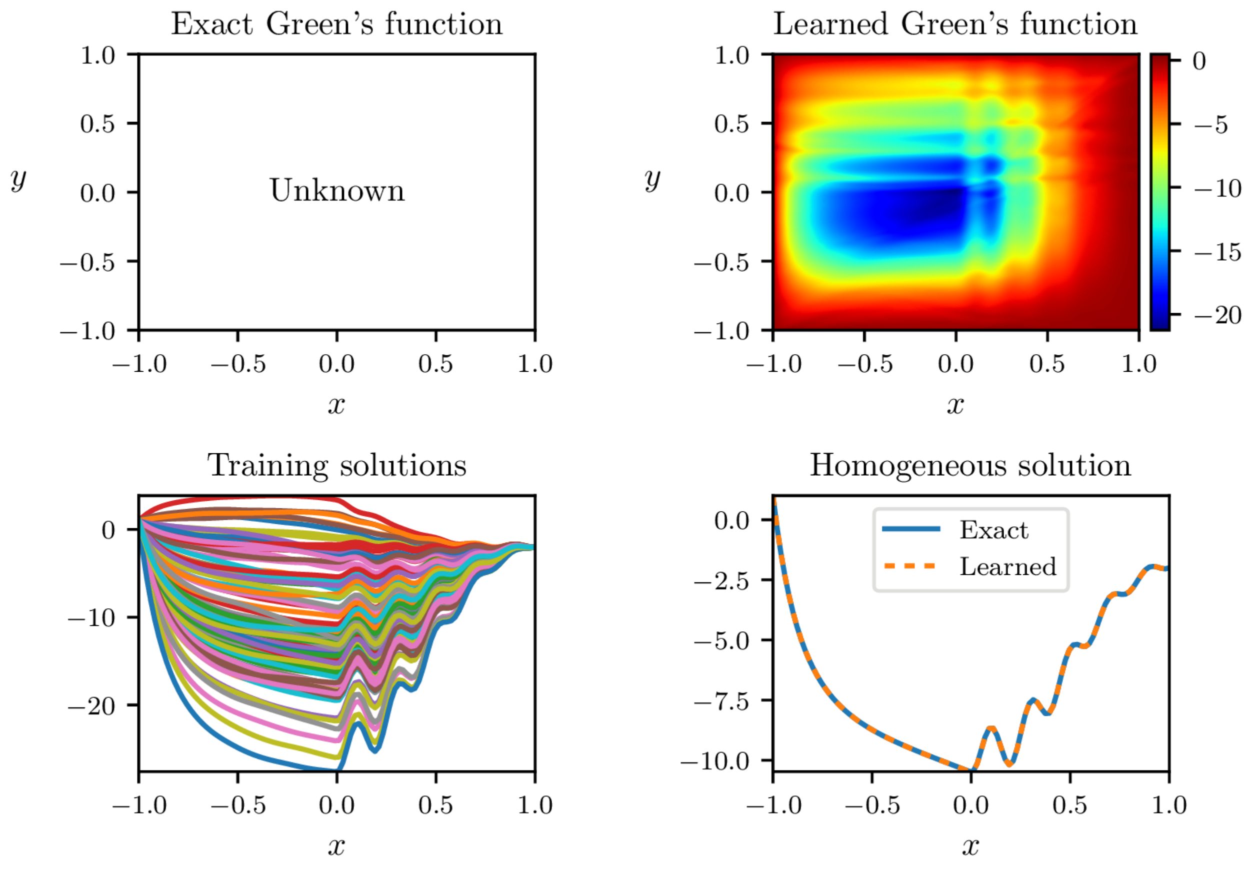
### 2.3.20 schrodinger



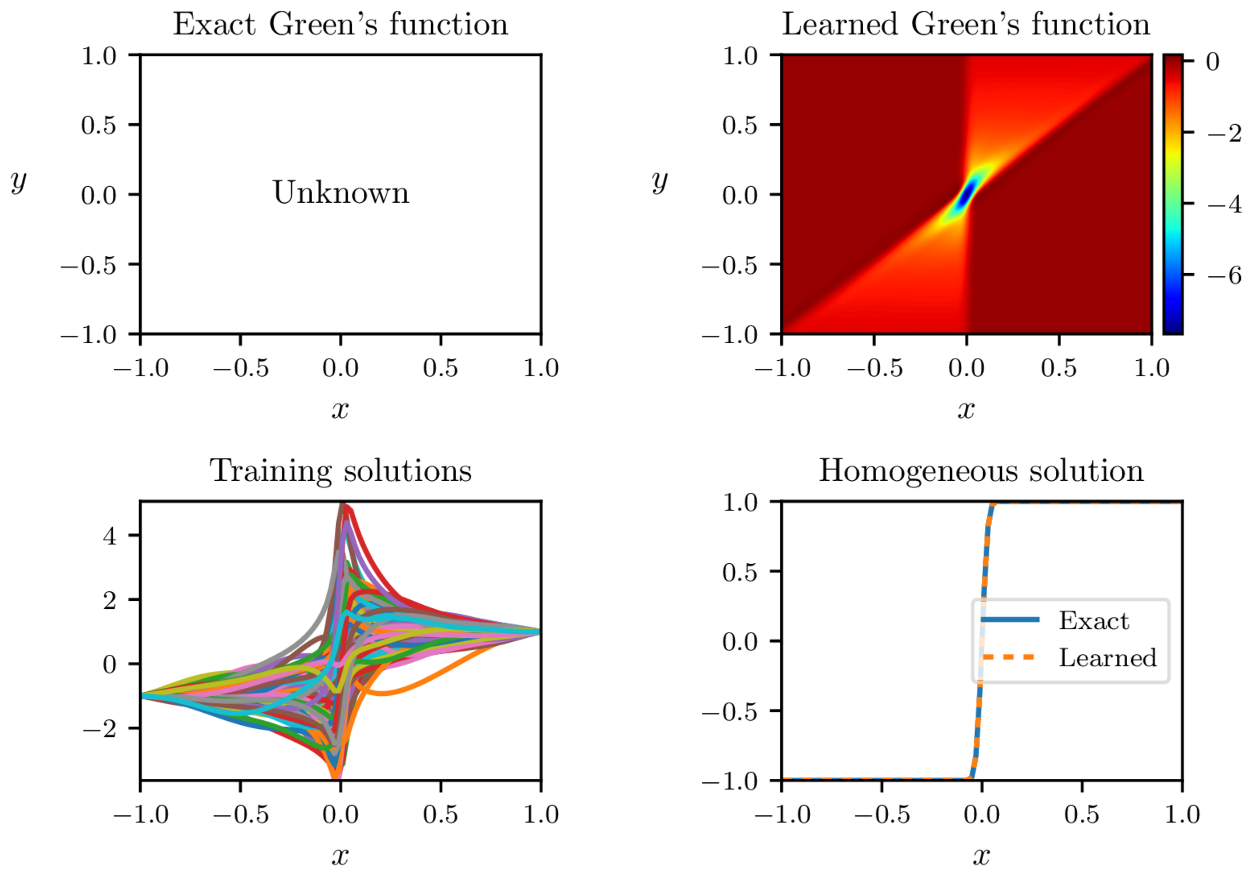
### 2.3.21 third_order



### 2.3.22 variable_coeffs



### 2.3.23 viscous_shock



## 2.4 Citation

Please cite the following papers if you are using GreenLearning.

- About GreenLearning:

```
@article{boulle2021mechanistic,
  title={Mechanistic understanding with Greens functions and deep learning},
  author={Boull{\'e}, Nicolas and Earl, Christopher J. and Townsend, Alex},
  journal={arXiv preprint arXiv:},
  year={2021}
}
```

- About Rational neural networks:

```
@inproceedings{boulle2020rational,
  title={Rational neural networks},
  author={Boull{\'e}, Nicolas and Nakatsukasa, Yuji and Townsend, Alex},
  booktitle = {Advances in Neural Information Processing Systems},
  volume = {33},
  pages = {14243--14253},
  year={2020},
```

(continues on next page)

(continued from previous page)

```
  url = {https://proceedings.neurips.cc/paper/2020/file/
↪a3f390d88e4c41f2747bfa2f1b5f87db-Paper.pdf}
}
```

CHAPTER

# THREE

# CODE DOCUMENTATION

This section contains the documentation of the classes and functions in the package.

## 3.1 greenlearning

### 3.1.1 greenlearning package

**class** greenlearning.**Model**(*G_network*, *U_hom_network*)

Bases: object

Create a model to learn Green's function from input-output data with deep learning.

Example:

```
 # Construct neural networks for G and homogeneous solution
G_network = gl.matrix_networks([2] + [50] * 4 + [1], "rational", (2,2))
U_hom_network = gl.matrix_networks([1] + [50] * 4 + [1], "rational", (2,))

# Define the model
model = gl.Model(G_network, U_hom_network)

# Train the model on the selected dataset in the path "examples/datasets/"
model.train("examples/datasets/", "ODE_system")

# Plot the results
model.plot_results()

# Close the session
model.sess.close()
```

**callback**(*loss*)

"Callback for optimizers: save the current value of the loss function.

**init_optimizer**()

Initialize the variables and optimizers.

**plot_results**()

Plot the learned Green's function and homogeneous solution.

**print_weights**()

Print all the trainable weights.

**save_loss**()

Save the loss function in a file after training.

**save_results**()
Save the Green's function evaluated at a grid in a csv file.

**train**(*example_path*, *example_name*)
Train the Green's function and homogeneous solution networks.

**class** greenlearning.**NeuralNetwork**(*layers*, *activation_name*)
Bases: `object`

Create a fully connected neural network with given number of layers and activation function.

Example:

```
gl.NeuralNetwork([2] + [50] * 4 + [1], "rational")
```

creates a rational neural network with 4 hidden layers of 50 neurons.

**evaluate**(*X*)
Evaluate the neural network at the array X.

**initialize_NN**()
Initialize the weights of the neural network.

**xavier_init**(*size*)
Initializer for weights and biases.

greenlearning.**matrix_networks**(*layers*, *activation*, *shape*)
Create a matrix of neural networks with the given parameters.

Example:

```
gl.matrix_networks([2] + [50] * 4 + [1], "rational", (2,1))
```

creates a matrix size 2 x 1 of rational networks with 4 hidden layers of 50 neurons.

## Subpackages

## greenlearning.utils package

## Submodules

## greenlearning.utils.backend module

Source: https://github.com/lululxvi/deepxde/blob/master/deepxde/backend.py

greenlearning.utils.backend.**backend**()
Returns the name and version of the current backend, e.g., ("tensorflow", 1.14.0).

**Returns** A `tuple` of the name and version of the backend GreenLearning is currently using.

**Return type** tuple

Example:

```
gl.utils.backend.backend()
>>> ("tensorflow", 1.15.0)
```

greenlearning.utils.backend.**is_tf_1**()
Check the version of Tensorflow.

### greenlearning.utils.config module

### greenlearning.utils.external_optimizer module

TensorFlow interface for third-party optimizers.

Code below is taken from https://github.com/tensorflow/tensorflow/blob/v1.15.2/tensorflow/contrib/opt/python/training/external_optimizer.py, because the `tf.contrib` module is not included in TensorFlow 2.

Another solution is using TensorFlow Probability, see the following references. But the following solution requires setting the weights before building the network and loss, which is not consistent with other optimizers in graph mode. A possible solution Could be adding a TFPOptimizerInterface similar to ScipyOptimizerInterface.

- https://www.tensorflow.org/probability/api_docs/python/tfp/optimizer/lbfgs_minimize
- https://github.com/tensorflow/probability/blob/master/tensorflow_probability/python/optimizer/lbfgs_test.py
- https://stackoverflow.com/questions/58591562/how-can-we-use-lbfgs-minimize-in-tensorflow-2-0
- https://stackoverflow.com/questions/59029854/use-scipy-optimizer-with-tensorflow-2-0-for-neural-network-training
- https://pychao.com/2019/11/02/optimize-tensorflow-keras-models-with-l-bfgs-from-tensorflow-probability/
- https://gist.github.com/piyueh/712ec7d4540489aad2dcfb80f9a54993
- https://github.com/pierremtb/PINNs-TF2.0/blob/master/utils/neuralnetwork.py

**class** greenlearning.utils.external_optimizer.**ExternalOptimizerInterface**(*loss, var_list=None, equalities=None, inequalities=None, var_to_bounds=None, **optimizer_kwargs*)

Bases: `object`

Base class for interfaces with external optimization algorithms. Subclass this and implement *_minimize* in order to wrap a new optimization algorithm. *ExternalOptimizerInterface* should not be instantiated directly; instead use e.g. *ScipyOptimizerInterface*. @@__init__ @@minimize

**minimize**(*session=None, feed_dict=None, fetches=None, step_callback=None, loss_callback=None, **run_kwargs*)

Minimize a scalar *Tensor*. Variables subject to optimization are updated in-place at the end of optimization. Note that this method does *not* just return a minimization *Op*, unlike *Optimizer.minimize()*; instead it actually performs minimization by executing commands to control a *Session*.

**Parameters**

- **session** – A *Session* instance.
- **feed_dict** – A feed dict to be passed to calls to *session.run*.
- **fetches** – A list of *Tensor`s to fetch and supply to `loss_callback* as positional arguments.

- **step_callback** – A function to be called at each optimization step; arguments are the current values of all optimization variables flattened into a single vector.
- **loss_callback** – A function to be called every time the loss and gradients are computed, with evaluated fetches supplied as positional arguments.
- ****run_kwargs** – kwargs to pass to *session.run*.

**class** greenlearning.utils.external_optimizer.**ScipyOptimizerInterface**(*loss, var_list=None, equalities=None, inequalities=None, var_to_bounds=None, **optimizer_kwargs*)

Bases: *greenlearning.utils.external_optimizer.ExternalOptimizerInterface*

Wrapper allowing *scipy.optimize.minimize* to operate a *tf.compat.v1.Session*.

Example:

```
vector = tf.Variable([7., 7.], 'vector')
# Make vector norm as small as possible.
loss = tf.reduce_sum(tf.square(vector))
optimizer = ScipyOptimizerInterface(loss, options={'maxiter': 100})
with tf.compat.v1.Session() as session:
    optimizer.minimize(session)
# The value of vector should now be [0., 0.].
```

Example with simple bound constraints:

```
vector = tf.Variable([7., 7.], 'vector')
# Make vector norm as small as possible.
loss = tf.reduce_sum(tf.square(vector))
optimizer = ScipyOptimizerInterface(loss, var_to_bounds={vector: ([1, 2], np.
↪infty)})
with tf.compat.v1.Session() as session:
    optimizer.minimize(session)
# The value of vector should now be [1., 2.].
```

Example with more complicated constraints:

```
vector = tf.Variable([7., 7.], 'vector')
# Make vector norm as small as possible.
loss = tf.reduce_sum(tf.square(vector))
# Ensure the vector's y component is = 1.
equalities = [vector[1] - 1.]
# Ensure the vector's x component is >= 1.
inequalities = [vector[0] - 1.]
# Our default SciPy optimization algorithm, L-BFGS-B, does not support
# general constraints. Thus we use SLSQP instead.
optimizer = ScipyOptimizerInterface(loss, equalities=equalities,␣
↪inequalities=inequalities, method='SLSQP')
with tf.compat.v1.Session() as session:
    optimizer.minimize(session)
# The value of vector should now be [1., 1.].
```

### greenlearning.utils.load_data module

`greenlearning.utils.load_data.`**`load_data`**(*model*, *example_path*, *example_name*)
    Load the training dataset.

### greenlearning.utils.plotting module

**`class`** `greenlearning.utils.plotting.`**`MathTextSciFormatter`**(*fmt='%1.2e'*)
    Bases: `matplotlib.ticker.Formatter`

    Format the axis of the figure.

`greenlearning.utils.plotting.`**`figsize`**(*scale*, *nplots=1*)
    Define the figure size.

`greenlearning.utils.plotting.`**`newfig`**(*width*, *nplots=1*)
    Create a new figure.

`greenlearning.utils.plotting.`**`savefig`**(*filename*, *crop=True*)
    Save the figure as a pdf file.

### greenlearning.utils.print_weights module

`greenlearning.utils.print_weights.`**`print_weights`**(*model*)
    Print all the trainable weights.

### greenlearning.utils.real module

Source: https://github.com/lululxvi/deepxde/blob/master/deepxde/real.py

**`class`** `greenlearning.utils.real.`**`Real`**(*precision*)
    Bases: `object`

    Set the float precision in numpy and Tensorflow.

    **`set_float32`**()

    **`set_float64`**()

    **`set_precision`**(*precision*)

### greenlearning.utils.save_results module

`greenlearning.utils.save_results.`**`save_results`**(*model*, *Green_slice=1*)
    Save the Green's function evaluated at a grid in a csv file. If the spatial dimension is equal to 2, Green_slice indicates the slice to save the Green's function'

### greenlearning.utils.tf_session module

`greenlearning.utils.tf_session.`**`open_tf_session`**`()`

Open a Tensorflow session.

### greenlearning.utils.visualization module

`greenlearning.utils.visualization.`**`input_data_slice`**`(`*model*, *Green_slice=1*`)`

Return evaluation points for the selected slice of the Green's function. Green_slice=1 : G(:,:,y1,y2), where y1, y2 are points in the middle of the domain. Green_slice=2 : G(:,x2,:,y2). Green_slice=3 : G(x1,x2,:,:), where x1, x2 are points in the middle of the domain. Green_slice=4 : G(x1,:,y1,:).

`greenlearning.utils.visualization.`**`plot_1d_results`**`(`*model*`)`

Plot the learned Green's function and homogeneous solution in 1D.

`greenlearning.utils.visualization.`**`plot_1d_systems`**`(`*model*`)`

Plot the learned Green's functions and homogeneous solutions for a system of PDEs in 1D.

`greenlearning.utils.visualization.`**`plot_2d_results`**`(`*model*`)`

Plot the learned Green's function and homogeneous solution in 2D.

`greenlearning.utils.visualization.`**`plot_2d_systems`**`(`*model*, *Green_slice=1*`)`

Plot the learned Green's functions and homogeneous solutions of a system of PDEs in 2D.

`greenlearning.utils.visualization.`**`plot_results`**`(`*model*`)`

Plot the learned Green's function and homogeneous solution in a pdf.

## Submodules

### greenlearning.activations module

`greenlearning.activations.`**`get`**`(`*identifier*, *weights*`)`

Return the activation function.

`greenlearning.activations.`**`initialize_weights`**`(`*identifier*`)`

Initialize the weights of the activation functions.

`greenlearning.activations.`**`rational`**`(`*x*, *weights*`)`

Define the rational activation function.

### greenlearning.loss_function module

**`class`** `greenlearning.loss_function.`**`loss_function`**`(`*G*, *N*`)`

Bases: `object`

Loss function for learning Green's functions and homogeneous solutions.

Inputs: matrices of neural networks G and N.

**`build`**`()`

Create Tensorflow placeholders and build the loss function.

**`feed_dict`**`(`*inputs_xU*, *inputs_xF*, *inputs_f*, *inputs_u*, *weights_x*, *weights_y*`)`

Construct a feed_dict to feed values to TensorFlow placeholders.

**`property outputs`**

## greenlearning.matrix_networks module

greenlearning.matrix_networks.**matrix_networks**(*layers*, *activation*, *shape*)

Create a matrix of neural networks with the given parameters.

Example:

```
gl.matrix_networks([2] + [50] * 4 + [1], "rational", (2,1))
```

creates a matrix size 2 x 1 of rational networks with 4 hidden layers of 50 neurons.

## greenlearning.model module

**class** greenlearning.model.**Model**(*G_network*, *U_hom_network*)

Bases: object

Create a model to learn Green's function from input-output data with deep learning.

Example:

```
 # Construct neural networks for G and homogeneous solution
G_network = gl.matrix_networks([2] + [50] * 4 + [1], "rational", (2,2))
U_hom_network = gl.matrix_networks([1] + [50] * 4 + [1], "rational", (2,))

# Define the model
model = gl.Model(G_network, U_hom_network)

# Train the model on the selected dataset in the path "examples/datasets/"
model.train("examples/datasets/", "ODE_system")

# Plot the results
model.plot_results()

# Close the session
model.sess.close()
```

**callback**(*loss*)

"Callback for optimizers: save the current value of the loss function.

**init_optimizer**()

Initialize the variables and optimizers.

**plot_results**()

Plot the learned Green's function and homogeneous solution.

**print_weights**()

Print all the trainable weights.

**save_loss**()

Save the loss function in a file after training.

**save_results**()

Save the Green's function evaluated at a grid in a csv file.

**train**(*example_path*, *example_name*)

Train the Green's function and homogeneous solution networks.

### greenlearning.neural_network module

**class** greenlearning.neural_network.**NeuralNetwork**(*layers*, *activation_name*)

Bases: object

Create a fully connected neural network with given number of layers and activation function.

Example:

```
gl.NeuralNetwork([2] + [50] * 4 + [1], "rational")
```

creates a rational neural network with 4 hidden layers of 50 neurons.

**evaluate**(*X*)

Evaluate the neural network at the array X.

**initialize_NN**()

Initialize the weights of the neural network.

**xavier_init**(*size*)

Initializer for weights and biases.

### greenlearning.quadrature_weights module

greenlearning.quadrature_weights.**get_weights**(*identifier*, *x*)

Get the type of quadrature weights associated to the numpy array x.

greenlearning.quadrature_weights.**trapezoidal**(*x*)

Trapezoidal weights for trapezoidal rule integration.

greenlearning.quadrature_weights.**uniform**(*x*)

Uniform weights for Monte-Carlo integration.

# PYTHON MODULE INDEX

# INDEX