Journal of Statistical Software

November 2024, Volume 111, Issue 6. doi: 10.18637/jss.v111.i06

pyStoNED: A Python Package for Convex Regression and Frontier Estimation

Sheng Dai
Zhongnan University of Economics and Law

Yu-Hsueh Fang
National Taiwan University

Chia-Yen Lee
National Taiwan University

Timo Kuosmanen
University of Turku

Abstract

Shape-constrained nonparametric regression is a growing area in econometrics, statistics, operations research, machine learning, and related fields. In the field of productivity and efficiency analysis, recent developments in multivariate convex regression and related techniques such as convex quantile regression and convex expectile regression have bridged the long-standing gap between the conventional deterministic-nonparametric and stochastic-parametric methods. Unfortunately, the heavy computational burden and the lack of a powerful, reliable, and fully open-access computational package have slowed down the diffusion of these advanced estimation techniques to the empirical practice. The purpose of the Python package **pyStoNED** is to address this challenge by providing a freely available and user-friendly tool for multivariate convex regression, convex quantile regression, convex expectile regression, isotonic regression, stochastic nonparametric envelopment of data, and related methods. This paper presents a tutorial of the **pyStoNED** package and illustrates its application, focusing on estimating frontier cost and production functions.

Keywords: multivariate convex regression, nonparametric least squares, frontier estimation, efficiency analysis, stochastic noise, Python.

1. Introduction

Early contributions to nonparametric regression by Hildreth (1954), Brunk (1955), and Grenander (1956) built exclusively on the convexity and monotonicity constraints of the regression function. However, extending these approaches from the univariate setting to the more general multivariate regression proved a vexing challenge. Since the development of an explicit piece-

wise linear characterization of the multivariate convex nonparametric least squares (CNLS) by Kuosmanen (2008), convex regression has attracted growing interest in econometrics, statistics, operations research, machine learning, and related fields (e.g., Magnani and Boyd 2009; Seijo and Sen 2011; Lim and Glynn 2012; Hannah and Dunson 2013; Mazumder, Choudhury, Iyengar, and Sen 2019; Bertsimas and Mundru 2021). The recent study by Yagi, Chen, Johnson, and Kuosmanen (2020) applies insights from convex regression to impose shape constraints on a local polynomial kernel estimator.

Convexity and monotonicity constraints are particularly relevant in the microeconomic applications where the duality theory of production and consumption directly implies certain monotonicity and convexity/concavity properties for many functions of interest (e.g., Afriat 1967, 1972; Varian 1982, 1984). For example, the cost function of a firm must be monotonic increasing and convex with respect to the input prices. Similar to the fact that a density function must be non-negative and its definite integral over the entire domain is equal to one, the cost function must satisfy the monotonicity and convexity properties implied by the theory, otherwise it is not really a cost function at all. The recent developments in the convex regression enable researchers to impose the concavity or convexity constraints implied by the theory to estimate the functions of interest without any parametric functional form assumptions.

In recent years, convex regression and related techniques have increasingly been utilized in the estimation of frontier cost and production functions in the field of productivity and efficiency analysis, a multidisciplinary field that is widely applied in such areas as agriculture, banking, education, environment, health care, energy, manufacturing, transportation, and utilities (e.g., Kuosmanen, Johnson, and Saastamoinen 2015; Johnson and Kuosmanen 2015). Traditionally, this field was divided into two competing paradigms: Data envelopment analysis (DEA; Charnes, Cooper, and Rhodes 1978) and stochastic frontier analysis (SFA; Aigner, Lovell, and Schmidt 1977; Meeusen, Van, and Broeck 1977). DEA is a deterministic, fully nonparametric approach whereas SFA is a probabilistic, fully parametric approach. To bridge the gap between these two paradigms, stochastic nonparametric envelopment of data (StoNED; Kuosmanen 2006; Kuosmanen and Kortelainen 2012) was proposed as a unified framework that combines the virtues of DEA and SFA, encompassing both approaches as its restricted special cases.

In practice, convex regression and StoNED are computationally demanding approaches, requiring a user to solve a mathematical programming problem subject to a large number of linear constraints. For example, the additive CNLS formulation by Kuosmanen (2008) is a quadratic programming (QP) problem, whereas the multiplicative logarithmic formulation first considered by Kuosmanen and Kortelainen (2012) requires solving a nonlinear programming (NLP) problem. Therefore, most empirical applications published thus far make use of commercial QP and NLP solvers, which can be coded using high-level mathematical computing languages such as GAMS (GAMS Development Corporation 2013) or MATLAB (The Mathworks, Inc. 2021). Johnson and Kuosmanen (2015) present detailed examples of how to calculate the basic CNLS and StoNED models in MATLAB and GAMS. Recently, the R (R Core Team 2024) package **Benchmarking** (Bogetoft and Otto 2010) provides a new function `StoNED()` to estimate CNLS/StoNED but limited to the additive CNLS QP formulation. Similarly, while **CVXPY** (Diamond and Boyd 2016) and **CVXOPT** (Andersen, Dahl, and Vandenberghe 2023) can be applied to solve CNLS, they are also restricted to the additive model only.

Model	**pyStoNED**	GAMS	MATLAB	**Benchmarking**
Convex regression				
Additive CNLS	✓	✓	✓	✓
Multiplicative CNLS	✓	✓		
Corrected CNLS	✓			
CQR/CER	✓			
CNLS with z variables	✓	✓	✓	
CQR/CER with z variables	✓			
CNLS with multiple outputs	✓	✓		
CQR/CER with multiple outputs	✓			
Isotonic CNLS	✓			
Isotonic CQR/CER	✓			
CNLS-G algorithm for CNLS	✓	✓	✓	
CNLS-G algorithm for CQR/CER	✓			
Frontier estimation				
StoNED with method of moments	✓	✓	✓	✓
StoNED with quasi-likelihood	✓	✓	✓	✓
StoNED with kernel deconvolution	✓			
Plotting				
One-input and one-output	✓			
Two-input and one-output	✓			

Table 1: Functionality comparison of different software packages.

The lack of a comprehensive, powerful, reliable, and fully open-access computational package for the CNLS, StoNED, and related methods has slowed down the diffusion of these techniques to the empirical practice, which still heavily relies on the simple DEA and SFA techniques that either assume away noise or rely on restrictive functional form assumptions. To lower the barrier for applied researchers and practitioners to apply more advanced techniques that help to relax unnecessarily restrictive assumptions, the **pyStoNED** package for Python (Van Rossum *et al.* 2021) was first introduced in April 2020 to prove a freely available and user-friendly tool for the multivariate CNLS and StoNED methods.

The **pyStoNED** package not only translates existing codes or tools to Python, but also provides many functionalities and modules that are not available in any other package or published code (see Table 1). Its latest edition also includes modules for convex quantile regression (CQR), convex expectile regression (CER), isotonic regression, penalized convex regression, and graphical illustration. It also facilitates efficiency measurement using the conventional DEA and free disposable hull (FDH) approaches. **pyStoNED** allows users to estimate these models in an open-access environment under a GPL-3.0 License. The project, including source code, internal data, notebook tutorials, and web documentation, is publicly available on the GitHub repository `https://github.com/ds2010/pyStoNED`.

This paper presents a tutorial of the **pyStoNED** package, briefly reviews the alternative models supported, and illustrates its application. We focus on the estimation of frontier cost and production functions, which currently forms the main application area of these techniques, emphasizing that the various modules in **pyStoNED** are directly applicable for

semi/nonparametric regression analysis in any other contexts as well (see, e.g., traffic flow modeling by Kriuchkov and Kuosmanen 2023). We emphasize that the **pyStoNED** package is an ongoing development by the users for the users: Further model specifications and methodological advances will be implemented and added to the **pyStoNED** package continuously.

The rest of this paper is organized as follows. Section 2 describes the basic setups of the **pyStoNED** package, and Section 3 introduces the structures of example data and the attributes of different models. Section 4 describes the first step of the StoNED model (e.g., CNLS estimation) to estimate the conditional mean and some other commonly used extensions. The Python code for the CNLS estimator and these extensions are included. Section 5 demonstrates the rest of the steps of the StoNED model and related codes. Section 6 illustrates how to implement the CNLS-G algorithm to calculate the CNLS estimator in **pyStoNED**. The plot of the estimated function can be found in Section 7. Section 8 concludes this paper. The list of acronyms is presented in Appendix A. Appendices B, C and D present a comparison of residuals for the same additive CNLS model solved by **pyStoNED**, GAMS and R/**Benchmarking**, respectively. Appendix E briefly describes the variables included in the four internal datasets.

2. Setup

2.1. Installation

The **pyStoNED** package supports Python 3.8 or later versions on Linux, macOS, and Windows, and is freely available on the Python package index (PyPI) at `https://pypi.org/project/pystoned`. The package can be installed either from PyPI or from GitHub using:[1]

1. `pip install pystoned`
2. `pip install -U git+https://github.com/ds2010/pyStoNED`

The **pyStoNED** package is built based on a few existing dependencies. It is worth highlighting that **Pyomo** (Bynum *et al.* 2021) is a full-featured high-level programming language that provides a rich set of supporting libraries to program CNLS, StoNED, and various extensions. The other dependencies are also essential for **pyStoNED**. Specifically, **NumPy** (Harris *et al.* 2020) and **pandas** (McKinney 2010) are used to import input-output data, manage data, and export the estimation results. **SciPy** (Virtanen *et al.* 2020) provides the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm to optimize the quasi-likelihood function. **Matplotlib** (Hunter 2007) is associated with the graphical illustration of the estimated functions.

2.2. Solvers

All models supported by **pyStoNED** are either additive or multiplicative models, depending on the specification of the error term. From the optimization perspective, the additive models are usually stated as QP problems except for the CQR model, which is a linear programming (LP) problem, whereas all multiplicative models are NLP problems.

[1] *Python for Non-Programmers* is recommended for beginners to learn and set up their Python environment, `https://wiki.python.org/moin/BeginnersGuide/NonProgrammers`.

To solve the relevant QP or NLP problems, external off-the-shelf solvers are required. In our experience, **CPLEX** (**CPLEX**, IBM ILOG 2009) and **MOSEK** (**MOSEK** ApS 2021) provide reliable and convenient platforms for solving the QP and LP problems in the present context. The NLP problem can be efficiently solved by **MINOS** (Murtagh and Saunders 2003) or **KNITRO** (Byrd, Nocedal, and Waltz 2006). With the help of **Pyomo**, all models supported by **pyStoNED** are computable by these off-the-shelf solvers.

Remote solver

pyStoNED interfaces with **Pyomo** to access the network-enabled optimization system (NEOS) server that freely provides a large number of academic solvers for solving the additive or multiplicative models remotely.[2] In this case, there is no need to install solvers and corresponding licenses on local machines. Here we have a model estimated by the remote solver

```
>>> model.optimize(email = "email@address")
```

Replace the argument `email@address` with your email address.[3] By default, the additive and multiplicative models will be solved by **MOSEK** and **KNITRO**, respectively. In addition, the users can check all the available solvers provided in NEOS via `get_remote_solvers()` and then select their preferred solver (e.g., `solver = "minos"`).

```
>>> from pystoned.utils.tools import get_remote_solvers
>>> print(get_remote_solvers())

['minto', 'couenne', 'lgo', 'minos', 'filmint', 'cbc', 'sbb', 'ipopt',
 'bonmin', 'lancelot', 'ooqp', 'knitro', 'miles', 'snopt', 'loqo',
 'path', 'dicopt', 'l-bfgs-b', 'baron', 'cplex', 'raposa', 'pathnlp',
 'octeract', 'scip', 'conopt', 'filter', 'nlpec', 'xpress', 'mosek',
 'minlp', 'gams-ampl', 'lindoglobal', 'alphaecp']

>>> model.optimize(email = "email@address", solver = 'minos')
```

The value of the option in `solver` could be any possible solver, which depends on the specific optimization problem and its availability. That is, there are a lot of different choices (see Table 2 for the commonly used setting), but the performance of solvers varies depending on the application.

Local solver

Pyomo also provides an application programming interface for **pyStoNED** to import the local solvers. In the **pyStoNED** package, **MOSEK** is attached as the internal dependency to solve the additive model. Note that the academic license for **MOSEK** is required to be installed; see The tutorial of **MOSEK** license installation is available at more tutorial on this installation.[4] After that, we can use the following command to calculate the additive models.

[2]NEOS server: State-of-the-art solvers for numerical optimization, https://neos-server.org/neos.

[3]As of January 2021, the NOES server requires a valid email address in all submissions; see NEOS Server FAQ https://neos-guide.org/content/FAQ#email.

[4]https://pystoned.readthedocs.io/en/latest/install/index.html#solver.

`solver =`	Remote	Local	QP	NLP	Recommended
`'cplex'`	✓		✓		
`'gurobi'`	✓		✓		
`'knitro'`	✓			✓	✓
`'mosek'`	✓	✓	✓		✓
`'minos'`	✓			✓	

Table 2: The commonly used option in `solver`.

```
>>> model.optimize(OPT_LOCAL)
```

The parameter `OPT_LOCAL` is added in the function `.optimize(...)` to indicate that the model is computed locally. By default, the additive model is solved by **MOSEK**. Furthermore, we can check if other solvers are preinstalled in the local environment. For example, in the following case, **CPLEX** has been installed on the machine using an academic license, and therefore, we can utilize it to solve the additive model.

```
>>> from pystoned.utils.tools import check_local_solver
>>> print(check_local_solver("cplex"))
```

```
True
```

```
>>> model.optimize(OPT_LOCAL, solver = 'cplex')
```

Overall, the remote solver through the NEOS server is highly recommended for all light computing jobs with no more than 500 observations. The local solver for calculating the multiplicative model will be supported in **pyStoNED** when a license for an appropriate NLP solver is available.

3. Data structures and dataset

3.1. Data structures

Data pre-processing is the first step to using the developed package, and thus, the user must prepare the dataset based on the data structures of **pyStoNED**. In all CNLS/StoNED models, two common vectors or matrix are required: Input variables $\boldsymbol{x}_i$ and output variables $\boldsymbol{y}_i$ for observed decision making unit (DMU) $i = 1, 2, \ldots, n$. For a model considering operational conditions and practices, contextual variables $\boldsymbol{z}_i$ can be used. The directional distance function (DDF) based models handle the multi-dimensional inputs and outputs with given directional vectors $\boldsymbol{g}^x$ and $\boldsymbol{g}^y$, respectively. The DDF based models also consider undesired outputs $\boldsymbol{b}_i$ and its directional vector $\boldsymbol{g}^b$. Table 3 summarizes the data structures used in all the CNLS/StoNED models.

3.2. Internal data

To illustrate the application of the **pyStoNED** package, four commonly used datasets are attached.

Symbol	Model	Description
$\boldsymbol{x}$	All models	Input variables
$\boldsymbol{y}$	All models	Output variables
$\boldsymbol{z}$	Contextual based models	Contextual variables
$\boldsymbol{b}$	DDF based models	Undesirable outputs
$\boldsymbol{g}^x$	DDF based models	The direction of inputs
$\boldsymbol{g}^y$	DDF based models	The direction of outputs
$\boldsymbol{g}^b$	DDF based models	The direction of undesirable outputs

Table 3: Data structures.

1. Finnish electricity distribution firms (`load_Finnish_electricity_firm`).

   The data of Finnish electricity distribution firms are collected from Kuosmanen (2012) and Kuosmanen, Saastamoinen, and Sipiläinen (2013). The data consist of seven variables: Three different expenditures are used as inputs (i.e., OPEX, CAPEX, and TOTEX);[5] Energy, Length, and Customers are considered as outputs; Further, PerUndGr is denoted as the contextual variable. Table 5 presents the description of this dataset.

2. GHG abatement cost of OECD countries (`load_GHG_abatement_cost`).

   The data on the greenhouse gas (GHG) abatement cost of OECD countries are provided by Kuosmanen, Zhou, and Dai (2020). The data contain two input variables (i.e., CPNK and HRSN), one good output variable (i.e., VALK), and one undesirable output variable (i.e., GHG) (see Table 6).

3. Data provided with Tim Coelli's **Frontier** 4.1 (`load_Tim_Coelli_frontier`).

   The classic 60-firm dataset attached in **Frontier** 4.1 (Coelli 1996) includes two input variables (i.e., capital and labour) and one output variable (i.e., output) (see Table 7).

4. Rice Production in the Philippines (`load_Philipines_rice_production`).

   The Rice Production in the Philippines dataset collected from Coelli, Rao, O'Donnell, and Battese (2005) consists of 17 different variables. The different variables can be organized into diversified combinations for target models (see Table 8).

These datasets can be imported through the module `dataset` of **pyStoNED**. The variables in Tables 5–8 are used as parameters of `dataset` to upload the input data. The following example demonstrates how to upload the input-output data of the Finnish electricity distribution firms dataset. We first import the dataset module using the function `load_Finnish_electricity_firm` (Line 1) and define the $\boldsymbol{x}$ and y according to the imported dataset (Line 2).[6] We then check the input and output data using the function `print()`.

[5]Note that TOTEX = OPEX + CAPEX. It is possible to use TOTEX as an aggregate input or model OPEX and CAPEX as two separate input variables.

[6]In addition to the presented example, we can download other datasets using the corresponding dataset module, e.g., downloading the GHG data: `from pystoned.dataset import load_GHG_abatement_cost`.

```
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'], z_select = ['PerUndGr'])

>>> print(data.x)

[[    75    878   4933]
     ...
 [   105    575   9084]]

>>> print(data.y)

[[  1612]
    ...
 [  1776]]

>>> print(data.z)

[[0.11]
  ...
 [0.59]]
```

The parameters `x_select`, `y_select`, and `z_select` in `load_Finnish_electricity_firm(x_select, y_select, z_select)` are used to select the inputs, outputs, and contextual variables, respectively. Note that the parameters in the module `dataset` can be defined according to the user's purpose. For example, if the target model only consists of two inputs (e.g., Energy and Customers) and one output (e.g., TOTEX), then Line 2 should be

```
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Customers'],
...   y_select = ['TOTEX'])
```

3.3. External data

In practice, the user's own dataset is the main input of **pyStoNED**. We present an example to show how to import the user's own data. Assume that Table 4 is the input-output data stored in an Excel file (e.g., `table1.xlsx`), the following code utilizes **pandas** to read the dataset from `table1.xlsx` and organize the data with **NumPy**. Input $\boldsymbol{x}$ is a matrix, and output y is an array.

In the following example, we first import the packages **NumPy** and **pandas** in Lines 1 and 2. Line 3 is used to read the `table.xlsx` file from the local disk, and the rest of the lines define the input and output variables. See more examples in Section 6.

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.read_excel("table1.xlsx")
>>> y = df['output']
```

id	output	input1	input2
1	120	10	55
2	80	30	49
⋮	⋮	⋮	⋮
100	90	25	72

Table 4: An example of user's own dataset.

```
>>> x1 = df['input1']
>>> x1 = np.asmatrix(x1).T
>>> x2 = df['input2']
>>> x2 = np.asmatrix(x2).T
>>> x = np.concatenate((x1, x2), axis = 1)

>>> print(y)

0     120
...
99     90

>>> print(x)

[[ 10  55]
...
 [ 25  72]]
```

4. Shape-constrained nonparametric regression

In this section, we review the commonly seen shape-constrained nonparametric regression models supported by **pyStoNED** and then illustrate how they can be solved and implemented in the developed package.[7]

Consider a standard multivariate, cross-sectional model in production economics

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i \tag{1}$$

where y_i is the output of the DMU i, $f : R_+^m \to R_+$ is the production (cost) function that characterizes the production (cost) technology, and $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})'$ denotes the input vector of unit i. To estimate the function f, one could resort to the parametric and nonparametric methods or neoclassical and frontier models, which are classified based on the specifications of f and error term ε (see Kuosmanen and Johnson 2010 for a review). In this paper, we assume certain axiomatic properties (e.g., monotonicity, concavity) instead of

[7]In addition, **pyStoNED** can also be used to solve other deterministic frontier models (e.g., DEA and FDH). See more DEA and FDH tutorials at https://pystoned.readthedocs.io/.

prespecified functional form for the function f and apply the following nonparametric methods to estimate the function f.

4.1. Convex nonparametric least squares

Additive CNLS model

Hildreth (1954) was the first to consider the nonparametric regression subject to monotonicity and concavity constraints in the case of a single input variable x. Afriat (1972) also proposes methods to impose convexity on estimating a production function. Kuosmanen (2008) extends Hildreth's approach to the multivariate setting with the multidimensional inputs $\boldsymbol{x}$, and refers to it as the CNLS. CNLS builds upon the assumption that the production function f belongs to a family of continuous, monotonic increasing, and globally concave (convex) functions, imposing the same production axioms as standard DEA (see further discussion in Kuosmanen and Johnson 2010). The two additive multivariate CNLS formulations are defined as

- Production function (i.e., regression function f is concave and increasing).

$$\begin{aligned} &\min_{\alpha,\boldsymbol{\beta},\varepsilon} \sum_{i=1}^{n} \varepsilon_i^2 && && (2)\\ &s.t. \quad y_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i && \forall i\\ &\qquad \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i && \forall i,j, \text{ and } i \neq j\\ &\qquad \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i \end{aligned}$$

- Cost function (i.e., regression function f is convex and increasing).

$$\begin{aligned} &\min_{\alpha,\boldsymbol{\beta},\varepsilon} \sum_{i=1}^{n} \varepsilon_i^2 && && (3)\\ &s.t. \quad y_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i && \forall i\\ &\qquad \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i \geq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i && \forall i,j, \text{ and } i \neq j\\ &\qquad \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i \end{aligned}$$

where α_i and $\boldsymbol{\beta}_i$ define the intercept and slope parameters of tangent hyperplanes that characterize the estimated piecewise linear frontier, respectively. The first constraint can be interpreted as a multivariate regression equation, the second constraint imposes convexity (concavity), and the third constraint imposes monotonicity. Problems (2) and (3) allow for variable returns to scale (VRS), and other specifications of returns to scale can be imposed by an additional constraint on the intercept term α_i. If $\alpha_i = 0$, problems (2) and (3) impose constant returns to scale (CRS). Note that both (2) and (3) are QP problems and hence can be solved by **MOSEK** or **CPLEX**.

The basic additive CNLS model can be estimated by **pyStoNED** using the module `CNLS(y, x, ...)` with the contextual variable `z` parameter set to `None` (default) and the type of model `cet` parameter set to `CET_ADDI` (additive model; default). The type of estimated function can be classified by setting the `fun` parameter to `FUN_PROD` (production function; default) or

`FUN_COST` (cost function). The returns to scale assumption can be specified by setting the `rts` parameter to `RTS_VRS` (VRS model; default) or `RTS_CRS` (CRS model). The estimated coefficients (e.g., $\hat{\alpha}_i$) can be displayed on the screen directly using `.display_alpha()` or stored in the memory using `.get_alpha()`.

We first present an example to solve the VRS production model.

```
>>> from pystoned import CNLS
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['OPEX', 'CAPEX'],
...   y_select = ['Energy'])
>>> model = CNLS.CNLS(y = data.y, x = data.x, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> model.optimize(OPT_LOCAL)
>>> model.display_alpha()
>>> model.display_beta()
>>> model.display_residual()
>>> alpha = model.get_alpha()
>>> beta = model.get_beta()
>>> residuals = model.get_residual()

alpha : alpha
  Size=89, Index=I
  Key : Lower : Value              : Upper : Fixed : Stale : Domain
    0 :  None : -22.93587954013432 :  None : False : False :  Reals
    ...
   88 :  None : -22.8603341852995  :  None : False : False :  Reals

beta : beta
  Size=178, Index=beta_index
  Key     : Lower : Value                : Upper : Fixed : Stale : Domain
   (0, 0) :   0.0 :  0.13585804486689262:  None : False : False :  Reals
    ...
  (88, 1) :   0.0 : 0.011417186366215866:  None : False : False :  Reals

epsilon : residual
  Size=89, Index=I
  Key : Lower : Value               : Upper : Fixed : Stale : Domain
    0 :  None :  -2.802404436894662 :  None : False : False :  Reals
    ...
   88 :  None : -0.8851559509243998 :  None : False : False :  Reals
```

In this example, Lines 1–3 import the CNLS module, parameter setting modules, and dataset module. Line 4 defines the input and output variables using the Finnish electricity distribution firm data. Lines 5–6 define the CNLS production model with a user-defined name (e.g., `model`) and solve the production model using the local **MOSEK** solver. Lines 7–9 directly display the

estimated coefficients (i.e., $\hat{\alpha}_i$, $\hat{\boldsymbol{\beta}}_i$, and $\hat{\varepsilon}_i$) on screen, and Lines 10–12 store the estimates in memory with a special variable name (e.g., alpha). See the online supplementary zip file for all the replicated scripts and results.

Appendix B presents the full estimated CNLS residuals that are equivalent to those in GAMS (cf. Appendix C). Note that the estimated alpha and beta coefficients in **pyStoNED** may be slightly different from those in GAMS because the optimal solution is generally not unique in terms of those coefficients, and hence there exist alternate optima (Kuosmanen 2008; Dai 2021).[8] The estimated residuals by **Benchmarking** are reported in Appendix D. The scatter plot in Figure 3 shows that the estimated residuals by **pyStoNED** and **Benchmarking** are similar, but not exactly the same. Further, the sum of squared residuals of **Benchmarking** is 0.17% higher than that of **pyStoNED** and GAMS, suggesting that **Benchmarking** fails to converge to the global optimum.

Multiplicative CNLS model

The Cobb-Douglas and translog functions are commonly used functional forms for the function f, where the error term ε affects output y in a multiplicative fashion. Thus, we next consider a multiplicative specification in the present nonparametric setting. Under the multiplicative error structure, model (1) is rephrased as

$$y_i = f(\boldsymbol{x}_i)\exp(\varepsilon_i) \tag{4}$$

Applying the log-transformation to (4), we have

$$\ln y_i = \ln f(\boldsymbol{x}_i) + \varepsilon_i \tag{5}$$

To estimate (5), we reformulate the additive production model (2) and obtain the following log-transformed CNLS formulation.

$$\begin{aligned} \min_{\alpha,\boldsymbol{\beta},\phi,\varepsilon} \; & \sum_{i=1}^{n} \varepsilon_i^2 \\ s.t. \quad & \ln y_i = \ln(\phi_i + 1) + \varepsilon_i && \forall i \\ & \phi_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i - 1 && \forall i \\ & \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i && \forall i, j, \text{ and } i \neq j \\ & \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i \end{aligned} \tag{6}$$

where $\phi_i + 1$ is the CNLS estimator of $\mathsf{E}[y_i \mid \boldsymbol{x}_i]$. The value of one is added here to ensure that the computational algorithms do not take the logarithm of zero. The first equality can be interpreted as the log-transformed regression equation (using the natural logarithm function $\ln(\cdot)$). The rest of the constraints are the same as those of the additive models. The use of ϕ_i allows the estimation of a multiplicative relationship between output and input while assuring convexity of the production possibility set in the original input-output space. Note that one could not apply the log transformation directly to the input data $\boldsymbol{x}$ due to the fact that

[8] The additive CNLS model can also be solved using other convex optimization tools such as **CVXPY** or **CVXOPT**. To apply these tools, a user must create QP matrices manually and then plug them into an off-the-shelf solver. For the interesting readers, see an additional example where the additive CNLS model is solved by **CVXOPT**, https://github.com/ds2010/CNLS-Python.

the piece-wise log-linear frontier does not satisfy the axiomatic property (i.e., concavity or convexity) of the function f. Since the multiplicative model (6) includes nonlinear constraints, we need to use NLP solvers such as **MINOS** and **KNITRO**.

We next demonstrate the estimation of multiplicative cost function under VRS and CRS. Let the type of model `cet` parameter be `CET_MULT` (multiplicative model). Note that the following NLP models are remotely solved by **KNITRO** via the NEOS server.

```
>>> from pystoned import CNLS
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RTS_CRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
>>> model1 = CNLS.CNLS(y = data.y, x = data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model1.optimize('email@address')
>>> model2 = CNLS.CNLS(y = data.y, x = data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_CRS)
>>> model2.optimize('email@address')
>>> model1.display_residual()
>>> model2.display_residual()
```

4.2. Convex quantile and expectile regression

Quantile regression estimates the conditional median or other quantiles of the response variable (Koenker and Bassett 1978; Koenker 2005), whereas the CNLS estimator focuses on the conditional mean $\mathsf{E}[y_i \mid \boldsymbol{x}_i]$. In this section, we extend CNLS to CQR (Wang, Wang, Dang, and Ge 2014) and CER (Kuosmanen *et al.* 2015), see Kuosmanen and Zhou 2021 and Dai, Kuosmanen, and Zhou 2023a for further elaboration of both CQR and CER. Note that both quantile and expectile estimators are generally more robust to outliers and heteroscedasticity than the conditional mean.

Convex quantile regression

Given a pre-specified quantile $\tau \in (0, 1)$, the CQR model is formulated as

$$\begin{aligned} \min_{\alpha,\boldsymbol{\beta},\varepsilon^+,\varepsilon^-} \; & \tau \sum_{i=1}^{n} \varepsilon_i^+ + (1-\tau) \sum_{i=1}^{n} \varepsilon_i^- & & (7) \\ s.t. \quad & y_i = \alpha_i + \boldsymbol{\beta}_i^{'} \boldsymbol{x}_i + \varepsilon_i^+ - \varepsilon_i^- & \forall i \\ & \alpha_i + \boldsymbol{\beta}_i^{'} \boldsymbol{x}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'} \boldsymbol{x}_i & \forall i, j, \text{ and } i \neq j \\ & \boldsymbol{\beta}_i \geq \mathbf{0} & \forall i \\ & \varepsilon_i^+ \geq 0, \; \varepsilon_i^- \geq 0 & \forall i \end{aligned}$$

where the quantile τ splits the observations $100 \cdot \tau\%$ above and $100 \cdot (1-\tau)\%$ below, and ε_i^+ and ε_i^- denote the two non-negative components. The objective function in (7) minimizes the asymmetric absolute deviations from the function rather than the symmetric quadratic deviations. The last set of constraints is the sign constraint of the error terms. The other

constraints are the same as those of the CNLS problem (2). See Dai (2021), Dai *et al.* (2023a), and Dai, Kuosmanen, and Zhou (2023b) for more recent developments on CQR models.

Convex expectile regression

Convex quantile regression (7) may suffer from non-uniqueness due to that problem (7) is an LP problem (Kuosmanen *et al.* 2015). To address this problem, Kuosmanen *et al.* (2015) propose a CER approach, where a quadratic objective function is used to ensure unique estimates of the quantile functions. Consider the following QP problem

$$\begin{aligned} \min_{\alpha,\boldsymbol{\beta},\varepsilon^+,\varepsilon^-} \quad & \tilde{\tau}\sum_{i=1}^{n}(\varepsilon_i^+)^2 + (1-\tilde{\tau})\sum_{i=1}^{n}(\varepsilon_i^-)^2 && (8)\\ s.t. \quad & y_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i^+ - \varepsilon_i^- && \forall i \\ & \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i && \forall i,j, \text{ and } i \neq j \\ & \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i \\ & \varepsilon_i^+ \geq 0,\ \varepsilon_i^- \geq 0 && \forall i \end{aligned}$$

where the expectile $\tilde{\tau} \in (0,1)$ is not the same as the quantile τ, but it can be converted from/to the quantile τ (Dai *et al.* 2023a). See Dai, Zhou, and Kuosmanen (2020), Kuosmanen *et al.* (2020), and Kuosmanen and Zhou (2021) for the empirical applications of CER.

The alternative module `CQER` includes the function `CQR(y, x, ...)` that is designed to solve the CQR problem and the function `CER(y, x, ...)` to solve the CER problem. Therefore, we use the `CQER.CQR()` and `CQER.CER()` to define the CQR problem (see, e.g., Line 5 in the following example) and the CER problem, respectively. The other parameters' settings are similar to those in module `CNLS()`. To display the estimated ε_i^+ and ε_i^-, the functions `.display_positive_residual()` and `.display_negative_residual()` are designed in the new module `CQER`. The following additive CQR model is presented to estimate a quantile production function.

```
>>> from pystoned import CQER
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned import dataset as dataset
>>> data = dataset.load_GHG_abatement_cost(x_select = ['HRSN', 'CPNK',
...   'GHG'], y_select = ['VALK'])
>>> model = CQER.CQR(y = data.y, x = data.x, tau = 0.5, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> model.optimize(OPT_LOCAL)
>>> model.display_alpha()
>>> model.display_beta()
>>> model.display_positive_residual()
>>> model.display_negative_residual()
```

4.3. Contextual variables

A firm's productive performance to operate efficiently typically depends on operational conditions and practices, such as the production environment and firm-specific characteristics

(i.e., technology selection and managerial practices). Johnson and Kuosmanen (2011, 2012) refer to both variables that characterize operational conditions and practices as contextual variables.

- Contextual variables are often, though not always, external factors that are beyond the control of firms.
  - Examples: Competition, regulation, weather, location;
  - Policymakers may influence the operating environment.
- Contextual variables can also be internal factors.
  - Examples: Management practices, ownership;
  - A better understanding of the internal factors can help the firm improve performance.

By introducing the contextual variables $\boldsymbol{z}_i = (z_{i1}, z_{i2}, \ldots, z_{ir})'$, the multiplicative model (5) is reformulated as a partial log-linear model to take the operational conditions and practices into account.

$$\ln y_i = \ln f(\boldsymbol{x_i}) + \boldsymbol{\lambda}' \boldsymbol{z}_i + \varepsilon_i$$

where parameter vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_r)$ represents the marginal effects of contextual variables on output. All other variables maintain their previous definitions. Similarly, we can also introduce the contextual variables to the additive model. In this section, we consider the multiplicative production model as our starting point.

CNLS with z variables

Following Johnson and Kuosmanen (2011), we incorporate the contextual variables in the multiplicative CNLS model and redefine it as follows

$$\begin{aligned} & \min_{\alpha,\boldsymbol{\beta},\boldsymbol{\lambda},\varepsilon} \sum_{i=1}^{n} \varepsilon_i^2 & \\ s.t. \quad & \ln y_i = \ln(\phi_i + 1) + \boldsymbol{\lambda}' \boldsymbol{z}_i + \varepsilon_i & \forall i \\ & \phi_i = \alpha_i + \boldsymbol{\beta}_i' \boldsymbol{x}_i - 1 & \forall i \\ & \alpha_i + \boldsymbol{\beta}_i' \boldsymbol{x}_i \leq \alpha_j + \boldsymbol{\beta}_j' \boldsymbol{x}_i & \forall i, j, \text{ and } i \neq j \\ & \boldsymbol{\beta}_i \geq \mathbf{0} & \forall i \end{aligned} \tag{9}$$

Denote by $\hat{\boldsymbol{\lambda}}$ the coefficients of the contextual variables obtained as the optimal solution to the above nonlinear problem. Johnson and Kuosmanen (2011) examine the statistical properties of this estimator in detail, showing its unbiasedness, consistency, and asymptotic efficiency.

The contextual variables $\boldsymbol{z}$ have been integrated into the modules `CNLS()` and `CQER()`. Further, the function `.display_lamda()` is used to display the marginal effect of contextual variables. In the following example, we estimate a log-transformed cost function model with z variable. Note that the model specification is assumed to be CRS in this example.

```
>>> from pystoned import CNLS
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_CRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'], z_select = ['PerUndGr'])
>>> model = CNLS.CNLS(y = data.y, x = data.x, z = data.z,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_CRS)
>>> model.optimize('email@address')
>>> model.display_lamda()
```

CER with z variables

Following Kuosmanen, Tan, and Dai (2023), we can incorporate contextual variables in the multiplicative CER estimation. The reformulation of the CER model is

$$\begin{aligned}
\min_{\alpha,\boldsymbol{\beta},\boldsymbol{\lambda},\varepsilon^+,\varepsilon^-} \quad & \tilde{\tau}\sum_{i=1}^{n}(\varepsilon_i^+)^2 + (1-\tilde{\tau})\sum_{i=1}^{n}(\varepsilon_i^-)^2 && && (10)\\
s.t. \quad & \ln y_i = \ln(\phi_i + 1) + \boldsymbol{\lambda}^{'}\boldsymbol{z}_i + \varepsilon_i^+ - \varepsilon_i^- && \forall i\\
& \phi_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i - 1 && \forall i\\
& \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i && \forall i, j, \text{ and } i \neq j\\
& \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i\\
& \varepsilon_i^+ \geq 0,\ \varepsilon_i^- \geq 0 && \forall i
\end{aligned}$$

The following code is prepared to solve the CER model with z-variable. We now use the function `CQER.CER(y, x, ...)` to model CER with z-variable and assume expectile $\tilde{\tau} = 0.5$. In this example, we also estimate a CRS cost function.

```
>>> from pystoned import CQER
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_CRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'], z_select = ['PerUndGr'])
>>> model = CQER.CER(y = data.y, x = data.x, z = data.z, tau = 0.5,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_CRS)
>>> model.optimize('email@address')
>>> model.display_lamda()
```

4.4. Multiple outputs

CNLS with multiple outputs

The convex regression approaches reviewed above have been presented within the single output, multiple input framework. This section describes the CNLS/CQR/CER approaches that use the directional distance function (DDF) to model multiple-input multiple-output data.

The DDF is a functional representation of the technology, defined as (Chambers, Chung, and Färe 1996, 1998)

$$\vec{D}_T(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{g^x}, \boldsymbol{g^y}) = \sup\{\theta \mid (\boldsymbol{x} - \theta \boldsymbol{g^x}, \boldsymbol{y} + \theta \boldsymbol{g^y}) \in T\},$$

where $(\boldsymbol{g^x}, \boldsymbol{g^y})$ is a direction vector, and $T = \{(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \text{ can produce } \boldsymbol{y}\}$ is the production possibility set. The DDF must always satisfy translation property and homogeneity of degree -1 in the direction vector, and it inherits the monotonicity and convexity properties of T.

In the stochastic setting, the data generating process satisfies the assumptions introduced by Kuosmanen and Johnson (2017), then the value of the DDF is equal to the random error term

$$\vec{D}_T(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{g^x}, \boldsymbol{g^y}) = \varepsilon_i, \forall i.$$

To estimate DDF empirically, we solve the following QP problem

$$\begin{aligned} \min_{\alpha,\boldsymbol{\beta},\boldsymbol{\gamma},\varepsilon} & \sum_{i=1}^{n} \varepsilon_i^2 & & \\ s.t. \quad & \boldsymbol{\gamma}_i^{'} \boldsymbol{y}_i = \alpha_i + \boldsymbol{\beta}_i^{'} \boldsymbol{x}_i - \varepsilon_i & & \forall i \\ & \alpha_i + \boldsymbol{\beta}_i^{'} \boldsymbol{x}_i - \boldsymbol{\gamma}_i^{'} \boldsymbol{y}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'} \boldsymbol{x}_i - \boldsymbol{\gamma}_j^{'} \boldsymbol{y}_i & & \forall i, j, \text{ and } i \neq j \\ & \boldsymbol{\gamma}_i^{'} g^y + \boldsymbol{\beta}_i^{'} g^x = 1 & & \forall i \\ & \boldsymbol{\beta}_i \geq \mathbf{0}, \boldsymbol{\gamma}_i \geq \mathbf{0} & & \forall i \end{aligned} \tag{11}$$

In addition to the same notations as the CNLS estimator, we also introduce firm-specific coefficients $\boldsymbol{\gamma_i}$ that represent marginal effects of outputs to the DDF.

The first constraint defines the distance to the boundary of T as a linear function of inputs and outputs. The linear approximation is based on the supporting hyperplanes of T analogous to CNLS (2). The second set of constraints is the system of Afriat inequalities that impose global concavity. The third constraint is a normalization constraint that ensures the translation property. The last two constraints impose monotonicity in all inputs and outputs.

To perform the DDF models, **pyStoNED** includes the module `CNLSDDF()`, which also allows one to include undesirable outputs denoted by $\boldsymbol{b}$. We present the CNLS-DDF models in the following two examples with and without undesirable outputs. To apply the module `CNLSDDF(y, x, ...)`, we have to pre-define the directional vector for the parameters `gx`, `gb` (None; default), and `gy` (see Line 5). The module reports the estimates using `.display_alpha()`, `.display_beta()`, `.display_gamma()`, and `.display_residual()`.

```
>>> from pystoned import CNLSDDF
>>> from pystoned.constant import FUN_PROD, OPT_LOCAL
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['OPEX', 'CAPEX'],
...   y_select = ['Energy','Length','Customers'])
>>> model = CNLSDDF.CNLSDDF(y = data.y, x = data.x, b = None,
...   fun = FUN_PROD, gx = [1.0, 0.0], gb = None, gy = [0.0, 0.0, 0.0])
>>> model.optimize(OPT_LOCAL)
>>> model.display_alpha()
>>> model.display_beta()
>>> model.display_gamma()
>>> model.display_residual()
```

When considering undesirable outputs $\boldsymbol{b}$, CNLS-DDF (11) can be reformulated as

$$\begin{aligned}
\min_{\alpha,\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta},\varepsilon} & \sum_{i=1}^{n} \varepsilon_i^2 \\
s.t. \quad & \boldsymbol{\gamma}_i^{'}\boldsymbol{y}_i - \boldsymbol{\delta}_i^{'}\boldsymbol{b}_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i - \varepsilon_i & \forall i \\
& \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \delta_i^{'}\boldsymbol{b}_i - \boldsymbol{\gamma}_i^{'}\boldsymbol{y}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i + \delta_j^{'}\boldsymbol{b}_i - \boldsymbol{\gamma}_j^{'}\boldsymbol{y}_i & \forall i,j, \text{ and } i \neq j \\
& \boldsymbol{\gamma}_i^{'}g^y + \boldsymbol{\beta}_i^{'}g^x + \boldsymbol{\delta}_i^{'}g^b = 1 & \forall i \\
& \boldsymbol{\beta}_i \geq \mathbf{0}, \boldsymbol{\gamma}_i \geq \mathbf{0} & \forall i
\end{aligned}$$

where the coefficients $\boldsymbol{\delta_i}$ denote marginal effects of undesirable outputs to the DDF.

We can also model the undesirable outputs ($\boldsymbol{b}$) in the framework of DDF via the module `CNLSDDF(y, x, b, ...)` (Line 5; cf., the example above). The estimated coefficients can be displayed by using the function `.display_delta()`.

```
>>> from pystoned import CNLSDDF
>>> from pystoned.constant import FUN_PROD, OPT_LOCAL
>>> from pystoned import dataset as dataset
>>> data = dataset.load_GHG_abatement_cost()
>>> model = CNLSDDF.CNLSDDF(y = data.y, x = data.x, b = data.b,
...   fun = FUN_PROD, gx = [0.0, 0.0], gb = -1.0, gy = 1.0)
>>> model.optimize(OPT_LOCAL)
>>> model.display_delta()
```

CQR and CER with multiple outputs

Similar to CNLS with DDF, we present another two approaches integrating DDF to convex quantile/expectile regression by using the `CQERDDF()` module.[9]

- CQR-DDF model.

$$\begin{aligned}
\min_{\alpha,\boldsymbol{\beta},\boldsymbol{\gamma},\varepsilon^+,\varepsilon^-} & \tau\sum_{i=1}^{n} \varepsilon_i^+ + (1-\tau)\sum_{i=1}^{n} \varepsilon_i^- \\
s.t. \quad & \boldsymbol{\gamma}_i^{'}\boldsymbol{y}_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i^+ - \varepsilon_i^- & \forall i \\
& \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i - \boldsymbol{\gamma}_i^{'}\boldsymbol{y}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i - \boldsymbol{\gamma}_j^{'}\boldsymbol{y}_i & \forall i,j, \text{ and } i \neq j \\
& \boldsymbol{\gamma}_i^{'}g^y + \boldsymbol{\beta}_i^{'}g^x = 1 & \forall i \\
& \boldsymbol{\beta}_i \geq \mathbf{0}, \boldsymbol{\gamma}_i \geq \mathbf{0} & \forall i \\
& \varepsilon_i^+ \geq 0,\ \varepsilon_i^- \geq 0 & \forall i
\end{aligned}$$

[9]The undesirable outputs $\boldsymbol{b}$ can also be included, similar to the `CNLSDDF()` module, but are omitted here to keep the discussion and notations more concise.

- CER-DDF model.

$$\begin{aligned}
\min_{\alpha,\boldsymbol{\beta},\boldsymbol{\gamma},\varepsilon^+,\varepsilon^-} \quad & \tilde{\tau}\sum_{i=1}^{n}(\varepsilon_i^+)^2 + (1-\tilde{\tau})\sum_{i=1}^{n}(\varepsilon_i^-)^2 & \\
s.t. \quad & \boldsymbol{\gamma}_i^{'}\boldsymbol{y}_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i^+ - \varepsilon_i^- & \forall i \\
& \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i - \boldsymbol{\gamma}_i^{'}\boldsymbol{y}_i \leq \alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i - \boldsymbol{\gamma}_j^{'}\boldsymbol{y}_i & \forall i,j, \text{ and } i \neq j \\
& \boldsymbol{\gamma}_i^{'}g^y + \boldsymbol{\beta}_i^{'}g^x = 1 & \forall i \\
& \boldsymbol{\beta}_i \geq \mathbf{0}, \boldsymbol{\gamma}_i \geq \mathbf{0} & \forall i \\
& \varepsilon_i^+ \geq 0,\ \varepsilon_i^- \geq 0 & \forall i
\end{aligned}$$

Similar to the module `CNLSDDF()`, the module `CQERDDF()` uses the same parameters and is applied to estimate the CQR/CER-DDF model. The results can be reported using the functions that have been introduced in module `CQER.CQR()/.CER()`. For instance, the functions `.display_positive_residual()` and `.display_negative_residual()` are used to display the estimated residuals, respectively (Lines 7 and 8).

```
>>> from pystoned import CQERDDF
>>> from pystoned.constant import FUN_PROD, OPT_LOCAL
>>> from pystoned import dataset as dataset
>>> data = dataset.load_Finnish_electricity_firm(x_select = ['OPEX', 'CAPEX'],
...   y_select = ['Energy', 'Length', 'Customers'])
>>> model = CQERDDF.CQRDDF(y = data.y, x = data.x, b = None, tau = 0.9,
...   fun = FUN_PROD, gx = [1.0, 0.0], gb = None, gy = [0.0, 0.0, 0.0])
>>> model.optimize(OPT_LOCAL)
>>> model.display_positive_residual()
>>> model.display_negative_residual()
```

4.5. Relaxing convexity

Isotonic CNLS

This section introduces a variant of the CNLS estimator, isotonic CNLS that only imposes monotonicity. To relax the concavity assumption in CNLS estimation (i.e., estimating a production function), we rephrase the Afriat inequality constraint in problem (2).

Define the following binary matrix $\boldsymbol{P} = [p_{ij}]_{n\times n}$ to represent isotonicity (Keshvari and Kuosmanen 2013).

$$p_{ij} = \begin{cases} 1 & \text{if } x_i \preccurlyeq x_j \\ 0 & \text{otherwise} \end{cases}$$

We apply a simple enumeration method to define the elements of matrix $\boldsymbol{P}$ and then solve

the following QP problem

$$\begin{aligned}
&\min_{\alpha,\boldsymbol{\beta},\varepsilon} \sum_{i=1}^{n} \varepsilon_i^2 \\
&s.t. \quad y_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i && \forall i \\
&\qquad p_{ij}(\alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i) \leq p_{ij}(\alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i) && \forall i, j, \text{ and } i \neq j \\
&\qquad \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i
\end{aligned}$$

Note that the concavity constraints between units i and j are relaxed whenever $p_{ij} = 0$. If the $p_{ij} = 1$ for all i and j, then the above isotonic CNLS problem reduces to the CNLS problem.

To calculate the monotonic models, the **pyStoNED** package provides the modules `ICNLS(y, x, ...)` and `ICQER.ICQR(y, x, ...)`/`ICQER.ICER(y, x, ...)`, of which inherit the parameter settings from the modules `CNLS()` and `CQER()`. Therefore, the implementations of `ICNLS()` and `ICQER()` are similar to those of `CNLS()` and `CQER()`. Note that the matrix $\boldsymbol{P}$ is enumerated as an internal function in this module.

```
>>> from pystoned import ICNLS
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['OPEX', 'CAPEX'],
...   y_select = ['Energy'])
>>> model = ICNLS.ICNLS(y = data.y, x = data.x, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> model.optimize(OPT_LOCAL)
>>> model.display_residual()
```

Isotonic CQR and CER

Similar to isotonic CNLS, the isotonic CQR and CER approaches are defined as follows (Dai *et al.* 2023a)

- Isotonic CQR.

$$\begin{aligned}
&\min_{\alpha,\boldsymbol{\beta},\varepsilon^+,\varepsilon^-} \tau \sum_{i=1}^{n} \varepsilon_i^+ + (1-\tau) \sum_{i=1}^{n} \varepsilon_i^- \\
&s.t. \quad y_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i^+ - \varepsilon_i^- && \forall i \\
&\qquad p_{ij}(\alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i) \leq p_{ij}(\alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i) && \forall i, j, \text{ and } i \neq j \\
&\qquad \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i \\
&\qquad \varepsilon_i^+ \geq 0,\ \varepsilon_i^- \geq 0 && \forall i
\end{aligned}$$

- Isotonic CER.

$$\begin{aligned}
\min_{\alpha,\boldsymbol{\beta},\varepsilon^+,\varepsilon^-} \quad & \tilde{\tau}\sum_{i=1}^{n}(\varepsilon_i^+)^2 + (1-\tilde{\tau})\sum_{i=1}^{n}(\varepsilon_i^-)^2 \\
s.t. \quad & y_i = \alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i + \varepsilon_i^+ - \varepsilon_i^- && \forall i \\
& p_{ij}(\alpha_i + \boldsymbol{\beta}_i^{'}\boldsymbol{x}_i) \leq p_{ij}(\alpha_j + \boldsymbol{\beta}_j^{'}\boldsymbol{x}_i) && \forall i,j, \text{ and } i \neq j \\
& \boldsymbol{\beta}_i \geq \mathbf{0} && \forall i \\
& \varepsilon_i^+ \geq 0,\ \varepsilon_i^- \geq 0 && \forall i
\end{aligned}$$

These two isotonic CQR/CER models can be computed by using the function `ICQR(y, x, tau, ...)` or `ICER(y, x, tau, ...)` embedded in **pyStoNED**. The following example shows an isotonic CER model solved by the developed package.

```
>>> from pystoned import ICQER
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['OPEX', 'CAPEX'],
...   y_select = ['Energy'])
>>> model = ICQER.ICER(y = data.y, x = data.x, tau = 0.9, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> model.optimize(OPT_LOCAL)
>>> model.display_residual()
```

4.6. Corrected CNLS

To pave the way to the stochastic nonparametric frontier estimation to be examined in Section 5, we complete this section with a simple deterministic approach to frontier estimation.

Corrected convex nonparametric least squares (C^2NLS) (Kuosmanen and Johnson 2010) is a nonparametric extension of the corrected ordinary least squares (COLS), in which nonparametric least squares subject to monotonicity and concavity constraints replace the first-stage parametric ordinary least squares (OLS) regression. To estimate the frontier production function f using the C^2NLS estimator, we assume that f is a monotonic increasing and globally concave production function. The key difference to model (1) is that we now assume $\varepsilon_i \leq 0$ represent technical inefficiency. Otherwise, we maintain the assumption that the inefficiencies ε are uncorrelated with inputs $\boldsymbol{x}$.

Similar to COLS, C^2NLS includes two steps stated as follows:

- Estimate the residuals $\varepsilon_i^{\text{CNLS}}$ by solving the additive CNLS model (2).
- Obtain the shifted residuals $\hat{\varepsilon}_i^{\text{C2NLS}} = \varepsilon_i^{\text{CNLS}} - \max_j \varepsilon_j^{\text{CNLS}}$ such that $\hat{\varepsilon}_i^{\text{C2NLS}}$ satisfy $\hat{\varepsilon} \leq 0$ with 0 indicating efficient performance. Subsequently, we adjust the CNLS intercepts α_i as

$$\hat{\alpha}_i^{\text{C2NLS}} = \alpha_i^{\text{CNLS}} + \max_j \varepsilon_j^{\text{CNLS}}$$

  where α_i^{CNLS} is the optimal intercept for firm i in the above CNLS problem and $\hat{\alpha}_i^{\text{C2NLS}}$ is the estimated intercept term in the C^2NLS problem. Slope coefficients $\boldsymbol{\beta}_i$ for C^2NLS are obtained directly as the optimal solution to the CNLS problem.

To solve the C^{2}NLS model in **pyStoNED**, we introduce two new functions: `.get_adjusted_residual()` and `.get_adjusted_alpha()`. After completing the first-stage estimation, we use these two functions to obtain the adjusted residuals and intercept terms.

```
>>> from pystoned import CNLS
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> data = load_Finnish_electricity_firm(x_select = ['OPEX', 'CAPEX'],
...   y_select = ['Energy'])
>>> model = CNLS.CNLS(y = data.y, x = data.x, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> model.optimize(OPT_LOCAL)
>>> print(model.get_adjusted_residual())
>>> print(model.get_adjusted_alpha())
```

5. Stochastic nonparametric envelopment of data

Consider now the production model with a composite error structure

$$y_i = f(\boldsymbol{x}_i) + v_i - u_i \quad \forall i$$

In contrast to model (1), the error term ε_i here consists of the inefficiency term $u_i > 0$ and the stochastic noise term v_i ($\varepsilon_i = v_i - u_i$). For the cost function model, the error term $\varepsilon_i = v_i + u_i$. Combining virtues of SFA and DEA in a unified framework, Kuosmanen (2006) and Kuosmanen and Kortelainen (2012) propose the StoNED estimator to estimate the production/cost frontier and inefficiency. Specifically,

- Step 1: Estimating the conditional mean $\mathsf{E}[y_i \mid \boldsymbol{x}_i]$ using the CNLS estimator;
- Step 2: Estimating the expected inefficiency μ based on the residual $\varepsilon_i^{\text{CNLS}}$;
- Step 3: Estimating the StoNED frontier $\hat{f}^{\text{StoNED}}$ based on the $\hat{\mu}$;
- Step 4: Estimating firm-specific inefficiencies $\mathsf{E}[u_i \mid \varepsilon_i^{\text{CNLS}}]$.

Besides the CNLS estimator, we can apply other convex regression approaches such as isotonic CNLS and CNLS-DDF to estimate the conditional mean in the first step (see Keshvari and Kuosmanen 2013; Kuosmanen and Johnson 2017).

5.1. Estimating the expected inefficiency

After obtaining the residuals (e.g., $\hat{\varepsilon}_i^{\text{CNLS}}$) from the convex regression approaches, one can estimate the expected value of the inefficiency term $\mu = E(u_i)$. In practice, three commonly used methods are available to estimate the expected inefficiency μ: Method of moments (Aigner *et al.* 1977), quasi-likelihood estimation (Fan, Li, and Weersink 1996), and kernel deconvolution estimation (Hall and Simar 2002). We next briefly review these three approaches and demonstrate the application of **pyStoNED**; see Kuosmanen *et al.* (2015) for a more detailed theoretical introduction.

Method of moments

The method of moments requires some additional parametric distributional assumptions. Under the assumptions of half-normal inefficiency, $u_i \sim N^+(0, \sigma_u^2)$, and normal noise, $v_i \sim N(0, \sigma_v^2)$, the second and third central moments of the residual distribution are

$$M_2 = \left[\frac{\pi - 2}{\pi}\right]\sigma_u^2 + \sigma_v^2$$
$$M_3 = \left(\sqrt{\frac{2}{\pi}}\right)\left[1 - \frac{4}{\pi}\right]\sigma_u^2$$

The second and third central moments can be estimated based on the CNLS residuals as

$$\hat{M}_2 = \sum_{i=1}^{n}(\hat{\varepsilon}_i - \bar{\varepsilon})^2/n$$
$$\hat{M}_3 = \sum_{i=1}^{n}(\hat{\varepsilon}_i - \bar{\varepsilon})^3/n$$

Note that the third moment M_3 (which measures the skewness of the distribution) only depends on the standard deviation parameter σ_u of the inefficiency distribution. Thus, given the estimated $\hat{M}_3$ (which should be positive in the case of a cost frontier), we can estimate the parameters σ_u and σ_v by

$$\hat{\sigma}_u = \sqrt[3]{\frac{\hat{M}_3}{\left(\sqrt{\frac{2}{\pi}}\right)\left[1 - \frac{4}{\pi}\right]}}$$
$$\hat{\sigma}_v = \sqrt[2]{\hat{M}_2 - \left[\frac{\pi - 2}{\pi}\right]\hat{\sigma}_u^2}$$

To estimate the expected inefficiency ($\hat{\mu} = \hat{\sigma}_u\sqrt{2/\pi}$), we provide a module `StoNED()` that inherits the parameter setting from the module `CNLS()`. We estimate the conditional expected inefficiency through the example presented below. Following the stepwise procedure outlined above, we first utilize the module `CNLS()` to estimate the conditional mean $\mathsf{E}[y_i \mid \boldsymbol{x}_i]$ (i.e., Lines 5–6), then apply the module `StoNED(model)`, where the parameter `model` is defined as the name of the CNLS model, to decompose the estimated residuals (Line 7). We develop and import the parameter `RED_MOM` (Line 3) to decompose the residuals using the moment method. We finally resort to the function `.get_unconditional_expected_inefficiency(RED_MOM)` included in the module `StoNED()` to retrieve the expected inefficiency $\hat{\mu}$.

```
>>> from pystoned import CNLS, StoNED
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RED_MOM
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
```

```
>>> model = CNLS.CNLS(data.y, data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model.optimize('email@address')
>>> rd = StoNED.StoNED(model)
>>> print(rd.get_unconditional_expected_inefficiency(RED_MOM))
```

Quasi-likelihood estimation

Quasi-likelihood estimation is an alternative approach to decomposing σ_u and σ_v suggested by Fan *et al.* (1996). Given the shape of the CNLS curve, we apply the standard maximum likelihood method to estimate the parameters σ_u and σ_v. The quasi-likelihood function is formulated as

$$\ln L(\lambda) = -n \ln(\hat{\sigma}) + \sum \ln \Phi\left[\frac{-\hat{\varepsilon}_i \lambda}{\hat{\sigma}}\right] - \frac{1}{2\hat{\sigma}^2} \sum \hat{\varepsilon}_i^2$$

where

$$\hat{\varepsilon}_i = \hat{\varepsilon}_i^{\text{CNLS}} - (\sqrt{2}\lambda\hat{\sigma})/[\pi(1+\lambda^2)]^{1/2}$$
$$\hat{\sigma} = \left\{\frac{1}{n}\sum(\hat{\varepsilon}_i^{\text{CNLS}})^2 / \left[1 - \frac{2\lambda^2}{\pi(1+\lambda^2)}\right]\right\}.$$

Note that the quasi-likelihood function only consists of a single parameter λ (i.e., the signal-to-noise ratio $\lambda = \sigma_u/\sigma_v$).[10] The symbol Φ represents the cumulative distribution function of the standard normal distribution. In the **pyStoNED** package, we use the BFGS algorithm provided by **SciPy** to estimate the maximum likelihood function.

Since we apply the quasi-likelihood estimation to decompose the residuals, we need to import the parameter `RED_QLE` to the module `StoNED()`.

```
>>> from pystoned import CNLS, StoNED
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RED_QLE
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
>>> model = CNLS.CNLS(data.y, data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model.optimize('email@address')
>>> rd = StoNED.StoNED(model)
>>> print(rd.get_unconditional_expected_inefficiency(RED_QLE))
```

Kernel deconvolution estimation

While the method of moments and quasi-likelihood approaches require additional distributional assumptions for the inefficiency and noise terms, an alternative nonparametric estimation of the expected inefficiency μ is available by applying nonparametric kernel deconvolution,

[10] Note the notation difference between signal-to-noise ratio λ and the marginal effect of contextual variable λ in problems (9) and (10).

as proposed by Hall and Simar (2002). Note that the residual $\hat{\varepsilon}_i^{\text{CNLS}}$ is a consistent estimator of $e^o = \varepsilon_i + \mu$ (for production model). The density function of e^o is

$$\hat{f}_{e^o}(z) = (nh)^{-1} \sum_{i=1}^{n} K\left(\frac{z - e_i^o}{h}\right),$$

where $K(\cdot)$ is a compactly supported kernel, and h is a bandwidth. Hall and Simar (2002) show that the first derivative of the density function of the composite error term (f_ε') is proportional to that of the inefficiency term (f_u') in the neighborhood of μ. Therefore, a nonparametric estimator of expected inefficiency μ is obtained as

$$\hat{\mu} = \arg\max_{z \in C} (\hat{f}'_{e^o}(z)),$$

where C is a closed interval in the right tail of f_{e^o}.

We then import the parameter `RED_KDE` to decompose the estimated CNLS residuals.

```
>>> from pystoned import CNLS, StoNED
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RED_KDE
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
>>> model = CNLS.CNLS(data.y, data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model.optimize('email@address')
>>> rd = StoNED.StoNED(model)
>>> print(rd.get_unconditional_expected_inefficiency(RED_KDE))
```

5.2. Estimating the StoNED frontier

Having estimated the expected inefficiency $\hat{\mu}$ in Section 5.1, we can estimate the efficient frontier by adjusting the conditional mean function estimated by CNLS. In this subsection we briefly discuss the cases of the additive and multiplicative models under VRS and CRS. Given the fact that the function estimated by CNLS is only unique for the observed data points $(\boldsymbol{x}_i, y_i)$, we then follow Kuosmanen and Kortelainen (2012) to estimate the unique conditional mean function $\hat{g}_{\min}^{\text{CNLS}}(\boldsymbol{x})$ under VRS. Note that CRS is imposed by setting $\alpha = 0$.

$$\hat{g}_{\min}^{\text{CNLS}}(\boldsymbol{x}) = \min_{\alpha,\boldsymbol{\beta}} \{\alpha + \boldsymbol{\beta}' x \mid \alpha + \boldsymbol{\beta}' \boldsymbol{x}_i \geq \hat{g}^{\text{CNLS}}(\boldsymbol{x}_i), \forall i\}$$

where $\hat{g}^{\text{CNLS}}(\boldsymbol{x}_i)$ is the conditional mean function estimated by the CNLS estimator. We subsequently shift the conditional mean function $\hat{g}_{\min}^{\text{CNLS}}(\boldsymbol{x})$ to the frontier using

- Production function.
  - Additive model: $\hat{f}^{\text{StoNED}}(\boldsymbol{x}) = \hat{g}_{\min}^{\text{CNLS}}(\boldsymbol{x}) + \hat{\mu}$;
  - Multiplicative model: $\hat{f}^{\text{StoNED}}(\boldsymbol{x}) = \hat{g}_{\min}^{\text{CNLS}}(\boldsymbol{x}) \cdot \exp(\hat{\mu})$.
- Cost function.

- Additive model: $\hat{f}^{\text{StoNED}}(\boldsymbol{x}) = \hat{g}^{\text{CNLS}}_{\min}(\boldsymbol{x}) - \hat{\mu}$;
- Multiplicative model: $\hat{f}^{\text{StoNED}}(\boldsymbol{x}) = \hat{g}^{\text{CNLS}}_{\min}(\boldsymbol{x}) \cdot \exp(-\hat{\mu})$.

In the following example, we demonstrate how to obtain the StoNED frontier via the package. The method of moments is utilized to decompose the CNLS residuals and calculate $\hat{\mu}$, and we then use `.get_stoned()` to obtain the StoNED frontier (the last line).

```
>>> from pystoned import CNLS, StoNED
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RED_MOM
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
>>> model = CNLS.CNLS(data.y, data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model.optimize('email@address')
>>> rd = StoNED.StoNED(model)
>>> print(rd.get_stoned(RED_MOM))
```

5.3. Estimating firm-specific inefficiencies

If the efficient production or cost frontier is the object of interest, there is no need to proceed further to Step 4. Kuosmanen *et al.* (2015) emphasize that unbiased and consistent estimators of the frontier are available, however, there is no consistent method for estimating firm-specific efficiencies in the cross-sectional setting under noise. In essence, in this subsection we engage in the daunting task of trying to predict a realization of a random variable based on a single observation that is subject to noise. A reader should be duly warned that this is a highly speculative exercise.

Prediction of firm-specific inefficiency relies heavily on the parametric distributional assumptions. After estimating the expected inefficiency μ using the methods of the moment (MOM) or quasi-likelihood estimation (QLE), we can employ the JLMS estimator proposed by Jondrow, Lovell, Materov, and Schmidt (1982) to estimate the firm-specific inefficiencies (Johnson and Kuosmanen 2015). Under the assumption of a normally distributed error term and a half-normally distributed inefficiency term, Jondrow *et al.* (1982) formulate the conditional distribution of inefficiency u_i, given $\hat{\varepsilon}_i$, and proposes the inefficiency estimator as the conditional mean $\mathsf{E}[u_i \mid \hat{\varepsilon}_i]$.

The conditional expected inefficiency $\mathsf{E}[u_i \mid \hat{\varepsilon}_i]$ for production function and cost function are

- Production function.

$$\mathsf{E}[u_i \mid \hat{\varepsilon}_i] = \mu_{*i} + \sigma_* \left[\frac{\phi(-\mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] = \sigma_* \left[\frac{\phi(\hat{\varepsilon}_i \lambda/\sigma)}{1 - \Phi(\hat{\varepsilon}_i \lambda/\sigma)} - \frac{\hat{\varepsilon}_i \lambda}{\sigma} \right]$$

  where $\mu_{*i} = -\hat{\varepsilon}_i \sigma_u^2/\sigma^2$, $\sigma_*^2 = \sigma_u^2 \sigma_v^2/\sigma^2$, $\lambda = \sigma_u/\sigma_v$, and $\sigma^2 = \sigma_u^2 + \sigma_v^2$. The symbol ϕ is the standard normal density function, and the symbol Φ denotes the cumulative distribution function of the standard normal distribution.

- Cost function.

$$\mathsf{E}[u_i \mid \hat{\varepsilon}_i] = \mu_{*i} + \sigma_* \left[\frac{\phi(-\mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] = \sigma_* \left[\frac{\phi(\varepsilon_i \lambda/\sigma)}{1 - \Phi(-\varepsilon_i \lambda/\sigma)} + \frac{\varepsilon_i \lambda}{\sigma} \right]$$

  where $\mu_* = \varepsilon\sigma_u^2/\sigma^2$, $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$, $\lambda = \sigma_u/\sigma_v$, and $\sigma^2 = \sigma_u^2 + \sigma_v^2$.

The firm-level technical efficiency (TE) can be expressed based on the estimated conditional mean as

- Production function.
  - Multiplicative model: $\text{TE} = \exp(-\mathsf{E}[u_i \mid \varepsilon_i])$;
  - Additive model: $\text{TE} = \dfrac{y - \mathsf{E}[u_i \mid \varepsilon_i]}{y}$.
- Cost function.
  - Multiplicative model: $\text{TE} = \exp(\mathsf{E}[u_i \mid \varepsilon_i])$;
  - Additive model: $\text{TE} = \dfrac{y + \mathsf{E}[u_i \mid \varepsilon_i]}{y}$.

To calculate the firm-level technical efficiency, we resort to the function `.get_technical_inefficiency()`. We either set the parameter in function `.get_technical_inefficiency()` to `RED_MOM` (i.e., using the MOM approach to calculate the efficiency; Line 8) or to `RED_QLE` (i.e., using the quasi-likelihood estimation approach).

```
>>> from pystoned import CNLS, StoNED
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RED_MOM
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
>>> model = CNLS.CNLS(data.y, data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model.optimize('email@address')
>>> rd = StoNED.StoNED(model)
>>> print(rd.get_technical_inefficiency(RED_MOM))
```

6. CNLS-G algorithm

Since convex regression approaches impose the convexity (concavity) of the regression function using the Afriat inequality, the computation can become excessively expensive due to the $O(n^2)$ linear constraints. For example, if the data samples have 500 observations, the total number of linear constraints is equal to 250,000. To speed up computation, Lee, Johnson, Moreno-Centeno, and Kuosmanen (2013) propose a more efficient generic algorithm, CNLS-G, which uses the relaxed Afriat constraint set and iteratively adds violated constraints to the relaxed model as necessary. See further discussions in Lee *et al.* (2013).

To illustrate the CNLS-G algorithm, we follow Lee *et al.* (2013) to generate the input and output variables. In this section, we assume an additive production function with two-input and one-output, $y = x_1^{0.4} \times x_2^{0.4} + \varepsilon$. We randomly draw the inputs x_1 and x_2 from a uniform distribution, $x \sim U[1, 10]$, and the error term ε from a normal distribution, $\varepsilon \sim N(0, 0.7^2)$. We generate 500 artificial observations, estimate the CNLS problem (2) and the CER problem (8), and calculate the firm-level technical efficiency using the CNLS-G algorithm.

6.1. Solving CNLS model

We first compare the running time of the original CNLS and CNLS-G algorithm in the same computation environment. Line 1 imports the modules `CNLSG()` that is designed to perform the CNLS-G algorithm and `CNLS()`. Note that the module `CNLSG()` has the same parameters as the module `CNLS()`. Line 3 imports **NumPy** to provide multidimensional arrays and functions for linear algebra. We use **time** to count the running time for CNLS estimation (see Lines 4, 9, and 12) and employ the function `.get_runningtime()` to directly obtain the running time for CNLS-G algorithm (Line 15). To replicate the experiment, we set a random seed using `np.random.seed(0)`. Lines 6–8 generate the variables $\boldsymbol{x}$, ε, and y. `model1` and `model2` are the CNLS model and CNLS-G model, respectively. To count the number of constraints included in the CNLS-G algorithm, the module `CNLSG()` provides an internal function `.get_totalconstr()` (Line 17).

```
>>> from pystoned import CNLSG, CNLS
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> import numpy as np
>>> import time
>>> np.random.seed(0)
>>> x = np.random.uniform(low = 1, high = 10, size = (500, 2))
>>> epsilon = np.random.normal(loc = 0, scale = 0.7, size = 500)
>>> y = x[:, 0]**0.4 * x[:, 1]**0.4 + epsilon
>>> t1 = time.time()
>>> model1 = CNLS.CNLS(y, x, z = None, cet = CET_ADDI, fun = FUN_PROD,
...   rts = RTS_VRS)
>>> model1.optimize(OPT_LOCAL)
>>> CNLS_time = time.time() - t1
>>> model2 = CNLSG.CNLSG(y, x, z = None, cet = CET_ADDI, fun = FUN_PROD,
...   rts = RTS_VRS)
>>> model2.optimize(OPT_LOCAL)
>>> print("The running time with algorithm is ", model2.get_runningtime())
>>> print("The running time without algorithm is ", CNLS_time)
>>> print("The total number of constraints is ", model2.get_totalconstr())
```

6.2. Solving CER model

We next demonstrate a CER model solved by the CNLS-G algorithm prepared in module `CQERG()`. The other experimental settings are similar to those in Section 6.1. Note that the CNLS-G algorithm can also solve the CQR model via the function `CQERG.CQRG(y, x, ...)`.

```
>>> from pystoned import CQERG, CQER
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> import numpy as np
>>> import time
>>> np.random.seed(0)
>>> x = np.random.uniform(low = 1, high = 10, size = (500, 2))
>>> epsilon = np.random.normal(loc = 0, scale = 0.7, size = 500)
>>> y = x[:, 0]**0.4 * x[:, 1]**0.4 + epsilon
>>> tau = 0.5
>>> t1 = time.time()
>>> model1 = CQER.CER(y, x, tau, z = None, cet = CET_ADDI, fun = FUN_PROD,
...   rts = RTS_VRS)
>>> model1.optimize(OPT_LOCAL)
>>> CER_time = time.time() - t1
>>> model2 = CQERG.CERG(y, x, tau, z = None, cet = CET_ADDI, fun = FUN_PROD,
...   rts = RTS_VRS)
>>> model2.optimize(OPT_LOCAL)
>>> print("The running time with algorithm is ", model2.get_runningtime())
>>> print("The running time without algorithm is ", CER_time)
>>> print("The total number of constraints in CER model is ",
...   model2.get_totalconstr())
```

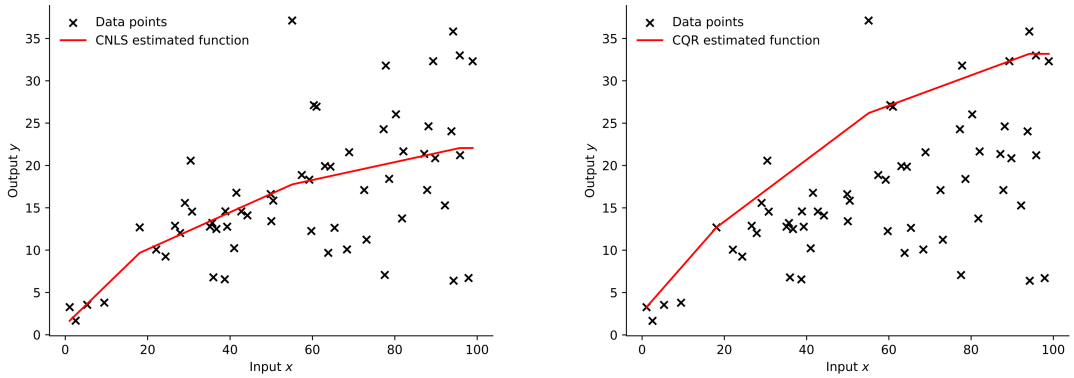
6.3. Calculating the expected inefficiency

We can apply the CNLS-G algorithm to calculate the unconditional expected inefficiency $\hat{\mu}$ more efficiently. In the first step of the `StoNED()` estimator, we use the module `CNLSG(y, x, ...)` as a substitute for the module `CNLS(y, x, ...)`.

```
>>> from pystoned import CNLSG, StoNED
>>> from pystoned.dataset import load_Finnish_electricity_firm
>>> from pystoned.constant import CET_MULT, FUN_COST, RTS_VRS, RED_MOM
>>> data = load_Finnish_electricity_firm(x_select = ['Energy', 'Length',
...   'Customers'], y_select = ['TOTEX'])
>>> model = CNLSG.CNLSG(data.y, data.x, z = None,
...   cet = CET_MULT, fun = FUN_COST, rts = RTS_VRS)
>>> model.optimize('email@address')
>>> rd = StoNED.StoNED(model)
>>> print(rd.get_unconditional_expected_inefficiency(RED_MOM))
```

7. Graphical illustration of estimated functions

To illustrate how the estimated regression function looks like, the **pyStoNED** package provides the functions `plot.plot2d()` and `plot.plot3d()` to plot two- and three-dimensional (2D and 3D) functions. As with the usage of the module `StoNED(y, x, ...)`, we first estimate the nonparametric regression such as CNLS, CQR, and isotonic CNLS and then apply the `plot`



(a) CNLS (b) CQR ($\tau = 0.9$)

Figure 1: CNLS and CQR estimates.

function to draw the figures. In this section, we use the internal data provided with Tim Coelli's **Frontier** 4.1 to demonstrate the process.

7.1. One-input and one-output

In the one-input and one-output case, we present two different estimated functions: The CNLS function and the CQR function. Therefore, we import the modules `CNLS` and `CQER` in Line 1 as the estimators, the plotting module `plot2d` (Line 2), and the example dataset (Line 4). Lines 6 and 7 and Lines 9 and 10 define and solve the CNLS and CQR models, respectively. Lines 8 and 11 are used to plot the estimated functions. There are four parameters in the module `plot2d(...)`: The first is the model's name, the second parameter `x_select` defines which the selected input x is, the third and last are the given names of the legend and generated picture, respectively. Figure 1 depicts the functions estimated by the CNLS and CQR model.

```
>>> from pystoned import CNLS, CQER
>>> from pystoned.plot import plot2d
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned.dataset import load_Tim_Coelli_frontier
>>> data = load_Tim_Coelli_frontier(x_select = ['labour'],
...   y_select = ['output'])
>>> CNLS_model = CNLS.CNLS(y = data.y, x = data.x, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> CNLS_model.optimize(OPT_LOCAL)
>>> plot2d(CNLS_model, x_select = 0, label_name = "CNLS estimated function",
...   fig_name = "CNLS_2d")
>>> CQR_model = CQER.CQR(y = data.y, x = data.x, tau = 0.5, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> CQR_model.optimize(OPT_LOCAL)
>>> plot2d(CQR_model, x_select = 0, label_name = "CQR estimated function",
...   fig_name = "CQR_2d")
```

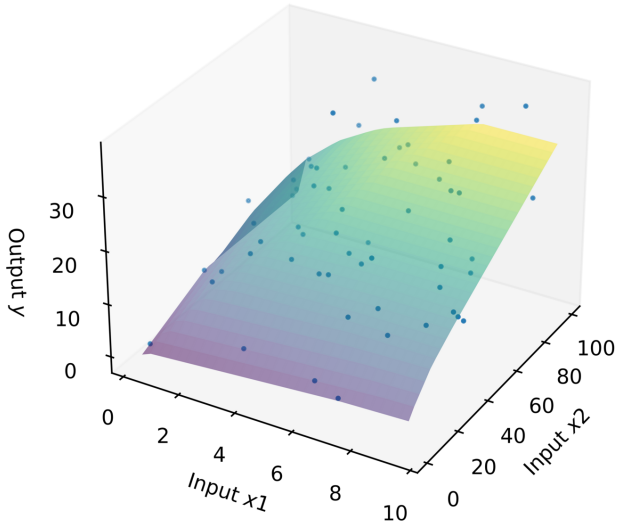


Figure 2: 3D plot of the piece-wise linear function estimated with CNLS.

7.2. Two-input and one-output

We first use the linear interpolation technique to obtain the 3D surface because CNLS estimates hyperplanes at the observed data points. The function `plot3d(model, x_select_1, x_select_2, fig_name = None, line_transparent = False, pane_transparent = False)` includes six parameters: The first is the name of the estimated model, the second and third define the selected input, the fourth is the name of the generated figure, the last two are the basic settings for the figure (False; default). We import function `plot3d(...)` in Line 2 and plot the figure in Line 8. Figure 2 presents the estimated 3D function.

```
>>> from pystoned import CNLS
>>> from pystoned.plot import plot3d
>>> from pystoned.constant import CET_ADDI, FUN_PROD, OPT_LOCAL, RTS_VRS
>>> from pystoned.dataset import load_Tim_Coelli_frontier
>>> data = load_Tim_Coelli_frontier(x_select = ['capital', 'labour'],
...   y_select = ['output'])
>>> CNLS_model = CNLS.CNLS(y = data.y, x = data.x, z = None,
...   cet = CET_ADDI, fun = FUN_PROD, rts = RTS_VRS)
>>> CNLS_model.optimize(OPT_LOCAL)
>>> plot3d(CNLS_model, x_select_1 = 0, x_select_2 = 1, fig_name = "CNLS_3d")
```

8. Conclusions

Convex regression and related methods provide an appealing way to impose shape constraints without making restrictive assumptions about the functional form. As these techniques are becoming increasingly popular, there is a great demand for a powerful, reliable, and fully open-access computational package. Several existing tools or packages have been developed and can be utilized in certain special cases. For example, R/**Benchmarking** only supports

additive CNLS/StoNED models, not CQR/CER or other more recent models. **CVXOPT** and **CVXPY** require that the user is able to manually construct the necessary QP matrices, which can be challenging for applied researchers and practitioners with limited experience in mathematical programming. Therefore, the developed **pyStoNED** package aims to fully address this need, providing a comprehensive set of functions for estimating CNLS, StoNED, and their numerous variants.

This paper has reviewed the models and specifications currently supported by **pyStoNED** and demonstrated its use with empirical examples taken from the literature on productivity and efficiency analysis. All modules are implemented in a fully open-access environment. We encourage the users to utilize **pyStoNED** to further develop their own packages for specific estimation purposes.

In the future, our plan is to include further extensions to **pyStoNED** on a continuous basis. One interesting avenue of ongoing research is to include additional penalty terms to the objective function of the CNLS/CQR/CER problems to alleviate overfitting and the curse of dimensionality (e.g., Lee and Cai 2020). More efficient computational algorithms such as the CNLS-A proposed by Dai (2021) are under active development and will be included in future editions of **pyStoNED**.

We hope that the **pyStoNED** package could contribute to raising the standards of empirical applications in productivity and efficiency analysis, and facilitate more meaningful and relevant empirical evidence that influence managerial and policy decisions. The basic idea of the StoNED approach is to enable applied researchers and practitioners of efficiency analysis to integrate existing tools and techniques from different domains such as econometrics, statistics, operational research, and machine learning into a logically consistent unified framework, and to facilitate further methodological development in a multi-disciplinary environment. The purpose of the **pyStoNED** package is to support this development.

While we have phrased this review and the **pyStoNED** modules in terms of cost and production functions, most of the modules are readily applicable to nonparametric regression analysis in any other context as well. We hope that the convex regression and related techniques could also prove useful in other application areas where shape constraints play an essential role. For example, optimization behavior also implies specific convexity constraints in the context of consumer demand analysis (see, e.g., Afriat 1967; Varian 1982).

Acknowledgments

We gratefully acknowledge financial support from the Foundation for Economic Education (Liikesivistysrahasto) [Nos. 180019, 190073, 210075], the HSE Support Foundation [No. 11–2290], and Ministry of Science and Technology, Taiwan [MOST108-2221–E–006–223–MY3].

References

Afriat SN (1967). "The Construction of Utility Functions from Expenditure Data." *International Economic Review*, **8**(1), 67–77. doi:10.2307/2525382.

Afriat SN (1972). "Efficiency Estimation of Production Functions." *International Economic Review*, **13**(3), 568–598. doi:10.2307/2525845.

Aigner D, Lovell CK, Schmidt P (1977). "Formulation and Estimation of Stochastic Frontier Production Function Models." *Journal of Econometrics*, **6**(1), 21–37. doi:10.1016/0304-4076(77)90052-5.

Andersen M, Dahl J, Vandenberghe L (2023). ***CVXOPT**: A Python Package for Convex Optimization. Version 1.3*. URL https://cvxopt.org.

Bertsimas D, Mundru N (2021). "Sparse Convex Regression." *INFORMS Journal on Computing*, **33**(1), 262–279. doi:10.1287/ijoc.2020.0954.

Bogetoft P, Otto L (2010). *Benchmarking with DEA, SFA, and R*. Springer-Verlag, New York. doi:10.1007/978-1-4419-7961-2.

Brunk HD (1955). "Maximum Likelihood Estimates of Monotone Parameters." *The Annals of Mathematical Statistics*, **26**(4), 607–616. doi:10.1214/aoms/1177728420.

Bynum ML, Hackebeil GA, Hart WE, Laird CD, Nicholson BL, Siirola JD, Watson JP, Woodruff DL (2021). ***Pyomo** – Optimization Modeling in Python*. 3rd edition. Springer-Verlag. doi:10.1007/978-3-030-68928-5.

Byrd RH, Nocedal J, Waltz RA (2006). "**KNITRO**: An Integrated Package for Nonlinear Optimization." In *Large-Scale Nonlinear Optimization*, pp. 35–59. Springer-Verlag, New York. doi:10.1007/0-387-30065-1_4.

Chambers RG, Chung Y, Färe R (1996). "Benefit and Distance Functions." *Journal of Economic Theory*, **70**(2), 407–419. doi:10.1006/jeth.1996.0096.

Chambers RG, Chung Y, Färe R (1998). "Profit, Directional Distance Functions, and Nerlovian Efficiency." *Journal of Optimization Theory and Applications*, **98**(2), 351–364. doi:10.1023/a:1022637501082.

Charnes A, Cooper WW, Rhodes E (1978). "Measuring the Efficiency of Decision Making Units." *European Journal of Operational Research*, **2**(6), 429–444. doi:10.1016/0377-2217(78)90138-8.

Coelli T (1996). "A Guide to **FRONTIER** Version 4.1: A Computer Program for Stochastic Frontier Production and Cost Function Estimation." *CEPA Working Paper 96/08*, University of New England. URL http://www.uq.edu.au/economics/cepa/frontier.php.

Coelli T, Rao DSP, O'Donnell CJ, Battese GE (2005). *An Introduction to Efficiency and Productivity Analysis*. Springer-Verlag, New York. doi:10.1007/b136381.

CPLEX, IBM ILOG (2009). "V12.1: User's Manual for **CPLEX**." *International Business Machines Corporation*, **46**(53), 157. doi:10.1007/978-3-662-62185-1_2.

Dai S (2021). "Variable Selection in Convex Quantile Regression: L_1-Norm or L_0-Norm Regularization?" *European Journal of Operational Research*, **305**(1), 338–355. doi:10.1016/j.ejor.2022.05.041.

Dai S, Kuosmanen T, Zhou X (2023a). "Generalized Quantile and Expectile Properties for Shape Constrained Nonparametric Estimation." *European Journal of Operational Research*, **310**(2), 914–927. doi:10.1016/j.ejor.2023.04.004.

Dai S, Kuosmanen T, Zhou X (2023b). "Non-Crossing Convex Quantile Regression." *Economics Letters*, **233**, 111396. doi:10.1016/j.econlet.2023.111396.

Dai S, Zhou X, Kuosmanen T (2020). "Forward-Looking Assessment of the GHG Abatement Cost: Application to China." *Energy Economics*, **88**, 104758. doi:10.1016/j.eneco.2020.104758.

Diamond S, Boyd S (2016). "**CVXPY**: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research*, **17**(83), 1–5. doi:10.1109/pyhpc.2016.009.

Fan Y, Li Q, Weersink A (1996). "Semiparametric Estimation of Stochastic Production Frontier Models." *Journal of Business & Economic Statistics*, **14**(4), 460–468. doi:10.2307/1392254.

GAMS Development Corporation (2013). *General Algebraic Modeling System (GAMS) Release 24.2.1*. Washington, DC. URL http://www.gams.com/.

Grenander U (1956). "On the Theory of Mortality Measurement." *Scandinavian Actuarial Journal*, **1956**(2), 125–153. doi:10.1080/03461238.1956.10414944.

Hall P, Simar L (2002). "Estimating a Change Point, Boundary, or Frontier in the Presence of Observation Error." *Journal of the American Statistical Association*, **97**(458), 523–534. doi:10.1198/016214502760047050.

Hannah LA, Dunson DB (2013). "Multivariate Convex Regression with Adaptive Partitioning." *Journal of Machine Learning Research*, **14**(66), 3153–3188. doi:10.1201/9780367816377-7.

Harris CR, Millman KJ, Van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, Van Kerkwijk MH, Brett M, Haldane A, ndez del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE (2020). "Array Programming with **NumPy**." *Nature*, **585**, 357–362. doi:10.1038/s41586-020-2649-2.

Hildreth C (1954). "Point Estimates of Ordinates of Concave Functions." *Journal of the American Statistical Association*, **49**(267), 598–619. doi:10.2307/2281132.

Hunter JD (2007). "**Matplotlib**: A 2D Graphics Environment." *Computing in Science & Engineering*, **9**(3), 90–95. doi:10.1109/mcse.2007.55.

Johnson AL, Kuosmanen T (2011). "One-Stage Estimation of the Effects of Operational Conditions and Practices on Productive Performance: Asymptotically Normal and Efficient, Root-n Consistent StoNEZD Method." *Journal of Productivity Analysis*, **36**, 219–230. doi:10.1007/s11123-011-0231-5.

Johnson AL, Kuosmanen T (2012). "One-Stage and Two-Stage DEA Estimation of the Effects of Contextual Vriables." *European Journal of Operational Research*, **220**(2), 559–570. doi:10.1016/j.ejor.2012.01.023.

Johnson AL, Kuosmanen T (2015). "An Introduction to CNLS and StoNED Methods for Efficiency Analysis: Economic Insights and Computational Aspects." In SC Ray, SC Kumbhakar, P Dua (eds.), *Benchmarking for Performance Evaluation: A Production Frontier Approach*, chapter 3, pp. 117–186. Springer-Verlag. doi:10.1007/978-81-322-2253-8_3.

Jondrow J, Lovell CK, Materov IS, Schmidt P (1982). "On the Estimation of Technical Inefficiency in the Stochastic Frontier Production Function Model." *Journal of Econometrics*, **19**(2–3), 233–238. doi:10.1016/0304-4076(82)90004-5.

Keshvari A, Kuosmanen T (2013). "Stochastic Non-Convex Envelopment of Data: Applying Isotonic Regression to Frontier Estimation." *European Journal of Operational Research*, **231**(2), 481–491. doi:10.1016/j.ejor.2013.06.005.

Koenker R (2005). *Quantile Regression*. Cambridge University Press, Cambridge. doi:10.1017/cbo9780511754098.

Koenker R, Bassett G (1978). "Regression Quantiles." *Econometrica*, **46**(1), 33–50. doi:10.2307/1913643.

Kriuchkov I, Kuosmanen T (2023). "Stochastic Nonparametric Estimation of the Density-Flow Curve." *arXiv 2305.17517*, arXiv.org E-Print Archive. doi:10.48550/arXiv.2305.17517.

Kuosmanen T (2006). "Stochastic Nonparametric Envelopment of Data: Combining Virtues of SFA and DEA in a Unified Framework." *MTT Discussion Paper 3/2006*, SSRN. doi:10.2139/ssrn.905758.

Kuosmanen T (2008). "Representation Theorem for Convex Nonparametric Least Squares." *Econometrics Journal*, **11**(2), 308–325. doi:10.1111/j.1368-423x.2008.00239.x.

Kuosmanen T (2012). "Stochastic Semi-Nonparametric Frontier Estimation of Electricity Distribution Networks: Application of the StoNED Method in the Finnish Regulatory Model." *Energy Economics*, **34**(6), 2189–2199. doi:10.1016/j.eneco.2012.03.005.

Kuosmanen T, Johnson AL (2010). "Data Envelopment Analysis as Nonparametric Least-Squares Regression." *Operations Research*, **58**(1), 149–160. doi:10.1287/opre.1090.0722.

Kuosmanen T, Johnson AL (2017). "Modeling Joint Production of Multiple Outputs in StoNED: Directional Distance Function Approach." *European Journal of Operational Research*, **262**(2), 792–801. doi:10.1016/j.ejor.2017.04.014.

Kuosmanen T, Johnson AL, Saastamoinen A (2015). "Stochastic Nonparametric Approach to Efficiency Analysis: A Unified Framework." In J Zhu (ed.), *Data Envelopment Analysis*, chapter 7, pp. 191–244. Springer-Verlag, New York. doi:10.1007/978-1-4899-7553-9_7.

Kuosmanen T, Kortelainen M (2012). "Stochastic Non-Smooth Envelopment of Data: Semi-Parametric Frontier Estimation Subject to Shape Constraints." *Journal of Productivity Analysis*, **38**, 11–28. doi:10.1007/s11123-010-0201-3.

Kuosmanen T, Saastamoinen A, Sipiläinen T (2013). "What Is the Best Practice for Benchmark Regulation of Electricity Distribution? Comparison of DEA, SFA and StoNED Methods." *Energy Policy*, **61**, 740–750. doi:10.1016/j.enpol.2013.05.091.

Kuosmanen T, Tan A, Dai S (2023). "Performance Analysis of English Hospitals during the First and Second Waves of the Coronavirus Pandemic." *Health Care Management Science*, **26**, 447–460. doi:10.1007/s10729-023-09634-7.

Kuosmanen T, Zhou X (2021). "Shadow Prices and Marginal Abatement Costs: Convex Quantile Regression Approach." *European Journal of Operational Research*, **289**(2), 666–675. doi:10.1016/j.ejor.2020.07.036.

Kuosmanen T, Zhou X, Dai S (2020). "How Much Climate Policy Has Cost for OECD Countries?" *World Development*, **125**, 104681. doi:10.1016/j.worlddev.2019.104681.

Lee CY, Cai JY (2020). "LASSO Variable Selection in Data Envelopment Analysis with Small Datasets." *Omega*, **91**, 102019. doi:10.1016/j.omega.2018.12.008.

Lee CY, Johnson AL, Moreno-Centeno E, Kuosmanen T (2013). "A More Efficient Algorithm for Convex Nonparametric Least Squares." *European Journal of Operational Research*, **227**(2), 391–400. doi:10.1016/j.ejor.2012.11.054.

Lim E, Glynn PW (2012). "Consistency of Multidimensional Convex Regression." *Operations Research*, **60**(1), 196–208. doi:10.1287/opre.1110.1007.

Magnani A, Boyd SP (2009). "Convex Piecewise-Linear Fitting." *Optimization and Engineering*, **10**, 1–17. doi:10.1007/s11081-008-9045-3.

Mazumder R, Choudhury A, Iyengar G, Sen B (2019). "A Computational Framework for Multivariate Convex Regression and Its Variants." *Journal of the American Statistical Association*, **114**(525), 318–331. doi:10.1080/01621459.2017.1407771.

McKinney W (2010). "Data Structures for Statistical Computing in Python." In S van der Walt, J Millman (eds.), *Proceedings of the 9th Python in Science Conference*, pp. 51–56. doi:10.25080/majora-92bf1922-00a.

Meeusen W, Van J, Broeck D (1977). "Efficiency Estimation from Cobb-Douglas Production Functions with Composed Error." *International Economic Review*, **18**(2), 435–444. doi:10.2307/2525757.

MOSEK ApS (2021). *The **MOSEK** Optimization Toolbox for Python Manual, Version 9.2.47*. URL https://docs.mosek.com/9.2/toolbox/.

Murtagh B, Saunders M (2003). ***MINOS** 5.51 User's Guide*. URL http://stanford.edu/group/SOL/guides/minos551.pdf.

R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL https://www.R-project.org/.

Seijo E, Sen B (2011). "Nonparametric Least Squares Estimation of a Multivariate Convex Regression Function." *The Annals of Statistics*, **39**(3), 1633–1657. doi:10.1214/10-aos852.

The Mathworks, Inc (2021). *MATLAB – The Language of Technical Computing, Version R2021a*. Natick. URL http://www.mathworks.com/.

Van Rossum G, *et al.* (2021). *Python Programming Language*. URL https://www.python.org/.

Varian HR (1982). "The Nonparametric Approach to Demand Analysis." *Econometrica*, **50**(4), 945–973. doi:10.2307/1912771.

Varian HR (1984). "The Nonparametric Approach to Production Analysis." *Econometrica*, **52**(3), 579–597. doi:10.2307/1913466.

Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, Van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, Van Mulbregt P, **SciPy** 10 Contributors (2020). "**SciPy** 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods*, **17**, 261–272. doi:10.1038/s41592-019-0686-2.

Wang Y, Wang S, Dang C, Ge W (2014). "Nonparametric Quantile Frontier Estimation Under Shape Restriction." *European Journal of Operational Research*, **232**(3), 671–678. doi:10.1016/j.ejor.2013.06.049.

Yagi D, Chen Y, Johnson AL, Kuosmanen T (2020). "Shape-Constrained Kernel-Weighted Least Squares: Estimating Production Functions for Chilean Manufacturing Industries." *Journal of Business & Economic Statistics*, **38**(1), 43–54. doi:10.1080/07350015.2018.1431128.

A. List of acronyms

COLS: Corrected ordinary least squares

C^{2}NLS: Corrected convex nonparametric least squares

CER: Convex expectile regression

CNLS: Convex nonparametric least squares

CNLS-G: Convex nonparametric least squares generic algorithm

CQR: Convex quantile regression

CRS: Constant returns to scale

DDF: Directional distance function

DEA: Data envelopment analysis

DMU: Decision-making unit

FDH: Free disposal hull

MOM: Method of moments

QLE: Quasi-likelihood estimation

ICNLS: Isotonic convex nonparametric least squares

SFA: Stochastic frontier analysis

StoNED: Stochastic nonparametric envelopment of data

StoNEZD: Stochastic semi-nonparametric envelopment of Z variables data

KDE: Kernel density estimation

VRS: Variable returns to scale

B. CNLS residuals according to pyStoNED

```
epsilon : residual
    Size=89, Index=I
    Key : Lower : Value               : Upper : Fixed : Stale : Domain
      0 :  None :  -2.802404436894662 :  None : False : False :  Reals
      1 :  None :  1.4140382715619921 :  None : False : False :  Reals
      2 :  None :  -22.22373494628377 :  None : False : False :  Reals
      3 :  None : -350.91098977887805 :  None : False : False :  Reals
      4 :  None : -13.699936032611063 :  None : False : False :  Reals
      5 :  None :   101.0148658439256 :  None : False : False :  Reals
      6 :  None : -28.872353594697955 :  None : False : False :  Reals
      7 :  None : -14.039725089008101 :  None : False : False :  Reals
      8 :  None :  -0.847450934771274 :  None : False : False :  Reals
      9 :  None :   56.89432734221239 :  None : False : False :  Reals
     10 :  None :  285.50688604635707 :  None : False : False :  Reals
     11 :  None :   679.3838747131822 :  None : False : False :  Reals
     12 :  None : -20.230010332395608 :  None : False : False :  Reals
     13 :  None :   -70.0079122391594 :  None : False : False :  Reals
```

```
14 :  None :   10.50653709011658 :  None : False : False :  Reals
15 :  None :   74.59735322137823 :  None : False : False :  Reals
16 :  None :  -6.538934579439136 :  None : False : False :  Reals
17 :  None : -30.076418126433722 :  None : False : False :  Reals
18 :  None : -40.134736174252836 :  None : False : False :  Reals
19 :  None : -27.777844860524397 :  None : False : False :  Reals
20 :  None :   48.75947590024222 :  None : False : False :  Reals
21 :  None :   87.00802351835063 :  None : False : False :  Reals
22 :  None :  22.480692094771772 :  None : False : False :  Reals
23 :  None :   23.95074675252465 :  None : False : False :  Reals
24 :  None : -2.1416800611444273 :  None : False : False :  Reals
25 :  None : -22.836592892039732 :  None : False : False :  Reals
26 :  None :  -37.86103508965881 :  None : False : False :  Reals
27 :  None : -351.07095371577884 :  None : False : False :  Reals
28 :  None :   66.85757452071438 :  None : False : False :  Reals
29 :  None : -21.702789546997877 :  None : False : False :  Reals
30 :  None :  -8.951288175416948 :  None : False : False :  Reals
31 :  None :  216.00597134650724 :  None : False : False :  Reals
32 :  None :  37.597367489307146 :  None : False : False :  Reals
33 :  None : -31.811466632750154 :  None : False : False :  Reals
34 :  None :  -23.78468977794573 :  None : False : False :  Reals
35 :  None :  -201.3412378050266 :  None : False : False :  Reals
36 :  None :  -69.29031473208599 :  None : False : False :  Reals
37 :  None :   6.268753820663875 :  None : False : False :  Reals
38 :  None :  3.7558132227129164 :  None : False : False :  Reals
39 :  None :  -74.60738872277068 :  None : False : False :  Reals
40 :  None : -1.9747815676415144 :  None : False : False :  Reals
41 :  None :   -11.9709086463711 :  None : False : False :  Reals
42 :  None :  -236.7462908485852 :  None : False : False :  Reals
43 :  None :   6.574582728891016 :  None : False : False :  Reals
44 :  None :  11.656885681175297 :  None : False : False :  Reals
45 :  None : -20.004027899272046 :  None : False : False :  Reals
46 :  None :    -67.214557123046 :  None : False : False :  Reals
47 :  None :  -5.239151984902829 :  None : False : False :  Reals
48 :  None :  -55.94693127471322 :  None : False : False :  Reals
49 :  None :   265.5141124871416 :  None : False : False :  Reals
50 :  None :  1.9241615487843404 :  None : False : False :  Reals
51 :  None :  -17.65079498452741 :  None : False : False :  Reals
52 :  None :   4.656309746806329 :  None : False : False :  Reals
53 :  None :   38.85435764023464 :  None : False : False :  Reals
54 :  None : -2.9033098200666387 :  None : False : False :  Reals
55 :  None :  349.52828099701924 :  None : False : False :  Reals
56 :  None :  163.99051800024677 :  None : False : False :  Reals
57 :  None :  35.001367028656844 :  None : False : False :  Reals
58 :  None :  28.392952260738326 :  None : False : False :  Reals
59 :  None :  -99.13214427853407 :  None : False : False :  Reals
60 :  None :  32.788798287561356 :  None : False : False :  Reals
```

```
 61 :  None :  -604.0398415704849 :  None : False : False :  Reals
 62 :  None :   197.1010340111934 :  None : False : False :  Reals
 63 :  None :  21.411063675293747 :  None : False : False :  Reals
 64 :  None : -26.755370561407986 :  None : False : False :  Reals
 65 :  None : -15.129740447971187 :  None : False : False :  Reals
 66 :  None :   29.38417964181255 :  None : False : False :  Reals
 67 :  None :  -128.2709894317968 :  None : False : False :  Reals
 68 :  None :  -16.44514183547261 :  None : False : False :  Reals
 69 :  None : -13.386266515443253 :  None : False : False :  Reals
 70 :  None : -293.86157104059725 :  None : False : False :  Reals
 71 :  None :  -2.550599367605116 :  None : False : False :  Reals
 72 :  None :   476.6195990080548 :  None : False : False :  Reals
 73 :  None :  -9.803577087510305 :  None : False : False :  Reals
 74 :  None :   32.08949015324251 :  None : False : False :  Reals
 75 :  None : -17.241339123458943 :  None : False : False :  Reals
 76 :  None :  114.76794160604845 :  None : False : False :  Reals
 77 :  None : -3.8514874302135453 :  None : False : False :  Reals
 78 :  None :   35.39620600887872 :  None : False : False :  Reals
 79 :  None : -62.937360855631766 :  None : False : False :  Reals
 80 :  None :   8.919365675503656 :  None : False : False :  Reals
 81 :  None :  349.39726801431743 :  None : False : False :  Reals
 82 :  None :  -80.69962474583377 :  None : False : False :  Reals
 83 :  None :  -604.7581448377796 :  None : False : False :  Reals
 84 :  None : -59.839712241581026 :  None : False : False :  Reals
 85 :  None :   6.718359188173217 :  None : False : False :  Reals
 86 :  None :   6.173657522415056 :  None : False : False :  Reals
 87 :  None :  -6.061079238934642 :  None : False : False :  Reals
 88 :  None : -0.8851559509243998 :  None : False : False :  Reals
```

C. CNLS residuals according to GAMS

```
----    160 VARIABLE e.L  error terms
1    -2.802,  2     1.414,  3   -22.224,  4  -350.911,  5   -13.700
6   101.015,  7   -28.872,  8   -14.040,  9    -0.847,  10   56.894
11  285.507,  12  679.384,  13  -20.230,  14  -70.008,  15   10.507
16   74.597,  17   -6.539,  18  -30.076,  19  -40.135,  20  -27.778
21   48.760,  22   87.008,  23   22.481,  24   23.951,  25   -2.142
26  -22.837,  27  -37.861,  28 -351.071,  29   66.858,  30  -21.703
31   -8.951,  32  216.006,  33   37.597,  34  -31.811,  35  -23.785
36 -201.341,  37  -69.290,  38    6.269,  39    3.756,  40  -74.607
41   -1.975,  42  -11.971,  43 -236.746,  44    6.575,  45   11.657
46  -20.004,  47  -67.215,  48   -5.239,  49  -55.947,  50  265.514
51    1.924,  52  -17.651,  53    4.656,  54   38.854,  55   -2.903
56  349.528,  57  163.991,  58   35.001,  59   28.393,  60  -99.132
61   32.789,  62 -604.040,  63  197.101,  64   21.411,  65  -26.755
```

```
66  -15.130,  67   29.384,  68 -128.271,  69  -16.445,  70  -13.386
71 -293.862,  72   -2.551,  73  476.620,  74   -9.804,  75   32.090
76  -17.241,  77  114.768,  78   -3.852,  79   35.396,  80  -62.937
81    8.919,  82  349.397,  83  -80.700,  84 -604.758,  85  -59.840
86    6.718,  87    6.174,  88   -6.061,  89   -0.885
```

D. CNLS residuals according to R/Benchmarking

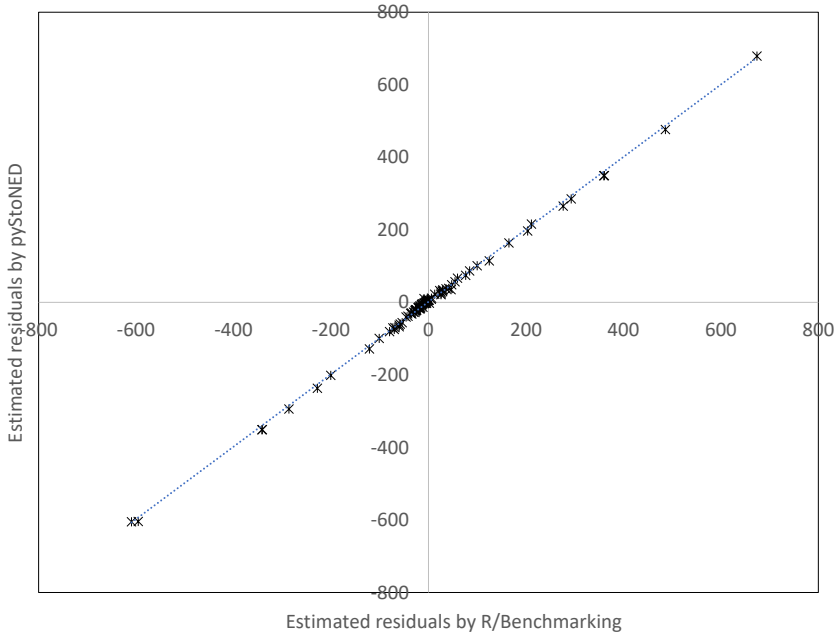


Figure 3: Scatterplot of residuals by **pyStoNED** and **Benchmarking**.

```
 [1,] -7.8886697   [2,]  -4.2032470  [3,] -26.3619713   [4,] -340.7346628
 [5,] -20.4750870  [6,]  99.8192026  [7,] -34.9414506   [8,] -16.0153732
 [9,] -6.6921578   [10,] 54.4126408  [11,] 292.9020295  [12,] 674.1823463
[13,] -24.5033276  [14,] -73.0074133 [15,] -9.0744802   [16,]  76.2853110
[17,] -13.4107689  [18,] -36.5828830 [19,] -46.0007323  [20,] -27.7717236
[21,]  47.7153749  [22,]  84.6132198 [23,]  12.8106176  [24,]  25.9149204
[25,]  -8.9327371  [26,] -27.3697694 [27,] -41.8614644  [28,] -341.8454225
[29,]  59.6833391  [30,] -24.1589554 [31,] -14.9332809  [32,] 211.3275027
[33,]  36.0900266  [34,] -34.4012906 [35,] -26.2801031  [36,] -200.4334730
[37,] -68.8716540  [38,]  0.7430369  [39,] -3.5393839   [40,] -70.1454171
[41,]   2.0062502  [42,] -19.2397146 [43,] -227.9417288 [44,]  1.3279244
[45,]   6.5195718  [46,] -26.4450121 [47,] -60.1349080  [48,] -13.4667633
[49,]  -53.8479333 [50,] 276.5003033 [51,] -5.0199937   [52,]  -18.2306460
[53,]  -0.1603049  [54,]  41.6619683 [55,]  -8.9830611  [56,] 360.9327664
[57,] 165.2316377  [58,]  29.7256506 [59,]  25.8260238  [60,] -100.9986163
```

```
[61,]  23.0478114  [62,]-595.6199337 [63,]  203.4132505 [64,]  25.6476906
[65,] -30.6974072  [66,] -21.2616803 [67,]  28.8204712  [68,] -121.1971133
[69,] -17.8872580  [70,] -11.4710720 [71,] -286.3945365 [72,] -3.7945771
[73,] 486.3441530  [74,] -17.3449996 [75,]  28.7225813  [76,] -16.7057597
[77,] 124.9659172  [78,] -11.6795149 [79,]  46.4608038  [80,] -60.0376607
[81,]   5.6434747  [82,] 359.8954805 [83,] -79.8715945  [84,] -609.4955011
[85,] -57.9114451  [86,]  -1.9436684 [87,]  -2.1260928  [88,] -14.3492663
[89,]  -5.8945399
```

E. Description of variables in the four internal datasets

Variable	Unit	Description
OPEX	Thousand Euro	Controllable operational expenditure
CAPEX	Thousand Euro	Total capital expenditure
TOTEX	Thousand Euro	Total expenditure
Energy	Gigawatt Hours	Weighted amount of energy transmitted
Length	Kilometer	Length of the network
Customers	Person	Customers connected to the network
PerUndGr	Percentage	Proportion of underground cabling

Table 5: Finnish electricity distribution firms.

Variable	Unit	Description
CPNK	Billion Euro^{2010}	Net capital stock
HRSN	Billion hours	Hours worked by total engaged
VALK	Billion Euro^{2010}	Value added
GHG	Million tons of CO_2 equivalents	Total GHG emissions

Table 6: GHG abatement cost of OECD countries.

Variable	Unit	Description
firm	Quantity	Firm ID
output	Quantity Index	Output quantity
capital	Quantity Index	Capital input
labour	Quantity Index	Labour input

Table 7: Data provided with Tim Coelli's **Frontier** 4.1.

Variable	Unit	Description
YEARDUM	Year	Time period
FMERCODE	Quantity	Farmer code
PROD	Tonnes	Tonnes of freshly threshed rice
AREA	Hactares	Area planted
LABOR	Mandays	Labour used
NPK	Kilogram	Fertiliser used
OTHER	Laspeyres index	Other inputs used
PRICE	Pesos/kilogram	Output price
AREAP	Pesos/hectare	Rental price of land
LABORP	Pesos/day	Labour price
NPKP	Pesos/kilogram	Fertiliser price
OTHERP	Implicit price index	Price of other inputs
AGE	Years	Age of the household head
EDYRS	Years	Education of the household head
HHSIZE	Quantity	Household size
NADULT	Quantity	Number of adults in the household
BANRAT	Percentage	Percentage of area classified as upland fields

Table 8: Rice production in the Philippines.

Affiliation:

Sheng Dai
School of Economics
Zhongnan University of Economics and Law
430073 Wuhan, China
Email: sheng.dai@zuel.edu.cn

Yu-Hsueh Fang, Chia-Yen Lee
Department of Information Management
National Taiwan University
Taipei City 106, Taiwan
Email: d11725001@ntu.edu.tw, chiayenlee@ntu.edu.tw

Timo Kuosmanen
Department of Economics
Turku School of Economics
University of Turku
FI-20014 Turun yliopisto, Finland
E-mail: timo.kuosmanen@utu.fi

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
November 2024, Volume 111, Issue 6 *Submitted:* 2021-09-25
doi:10.18637/jss.v111.i06 *Accepted:* 2024-04-20