# Package 'DSWE'

January 11, 2021

**Title** Data Science for Wind Energy

**Version** 1.5.1

**Description** Data science methods used in wind energy applications.
Current functionalities include creating a multi-dimensional power curve model,
performing power curve function comparison, and covariate matching.
Relevant works for the developed functions are:
funGP() - Prakash et al. (2020) <arxiv:2003.07899>,
AMK() - Lee et al. (2015) <doi:10.1080/01621459.2014.977385>,
tempGP() - Prakash et al. (2020) <arxiv:2012.01349>,
ComparePCurve() - Ding et al. (2020) <arxiv:2005.08652>,
All other functions - Ding (2019, ISBN:9780429956508).

**Depends** R (>= 3.5.0)

**License** MIT + file LICENSE

**URL** https://github.com/TAMU-AML/DSWE-Package,
https://aml.engr.tamu.edu/book-dswe/

**BugReports** https://github.com/TAMU-AML/DSWE-Package/issues

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**LinkingTo** Rcpp (>= 1.0.4.6) , RcppArmadillo (>= 0.9.870.2.0)

**Imports** Rcpp (>= 1.0.4.6) , matrixStats (>= 0.55.0) , FNN (>= 1.1.3) ,
KernSmooth (>= 2.23-16) , mixtools (>= 1.1.0), BayesTree (>=
0.3-1.4), gss (>= 2.2-2), e1071 (>= 1.7-3), stats (>= 3.5.0)

**NeedsCompilation** yes

**Author** Nitesh Kumar [aut],
Abhinav Prakash [aut],
Yu Ding [aut, cre],
Rui Tuo [ctb, cph]

**Maintainer** Yu Ding <yuding@tamu.edu>

**Repository** CRAN

**Date/Publication** 2021-01-11 10:30:07 UTC

# R topics documented:



---

AMK *Additive Multiplicative Kernel Regression*

---

## Description

An additive multiplicative kernel regression based on Lee et al. (2015).

## Usage

```
AMK(
  trainX,
  trainY,
  testX,
  bw = "dpi_gap",
  nMultiCov = 3,
  fixedCov = c(1, 2),
  cirCov = NA
)
```

## Arguments

| | |
|---|---|
| trainX | a matrix or dataframe of input variable values in the training dataset. |
| trainY | a numeric vector for response values in the training dataset. |
| testX | a matrix or dataframe of test input variable values to compute predictions. |

| | |
|---|---|
| bw | a numeric vector or a character input for bandwidth. If character, bandwidth computed internally; the input should be either 'dpi' or 'dpi_gap'. Default is 'dpi_gap'. See details for more information. |
| nMultiCov | an integer or a character input specifying the number of multiplicative covariates in each additive term. Default is 3 (same as Lee et al., 2015). The character inputs can be: 'all' for a completely multiplicative model, or 'none' for a completely additive model. Ignored if the number of covariates is 1. |
| fixedCov | an integer vector specifying the fixed covariates column number(s), default value is c(1,2). Ignored if nMultiCov is set to 'all' or 'none' or if the number of covariates is less than 3. |
| cirCov | an integer vector specifying the circular covariates column number(s) in trainX, default value is NA. |

## Details

This function is based on Lee et al. (2015). Main features are:

- Flexible number of multiplicative covariates in each additive term, which can be set using nMultiCov.
- Flexible number and columns for fixed covariates, which can be set using fixedCov. The default option c(1,2) sets the first two columns as fixed covariates in each additive term.
- Handling the data with gaps when the direct plug-in estimator used in Lee et al. fails to return a finite bandwidth. This is set using the option bw = 'dpi_gap' for bandwidth estimation.

## Value

a numeric vector for predictions at the data points in testX.

## References

Lee, Ding, Genton, and Xie, 2015, "Power curve estimation with multivariate environmental factors for inland and offshore wind farms," Journal of the American Statistical Association, Vol. 110, pp. 56-67, DOI:10.1080/01621459.2014.977385.

## Examples

```
data = data1
trainX = as.matrix(data[c(1:100),2])
trainY = data[c(1:100),7]
testX = as.matrix(data[c(101:110),2])
AMK_prediction = AMK(trainX, trainY, testX, bw = 'dpi_gap', cirCov = NA)
```

## BayesTreePCFit *Tree based power curve estimate*

### Description

Tree based power curve estimate

### Usage

```
BayesTreePCFit(trainX, trainY, testX, nTree = 50)
```

### Arguments

| | |
|---|---|
| trainX | a matrix or dataframe to be used in modelling |
| trainY | a numeric or vector as a target |
| testX | a matrix or dataframe, to be used in computing the predictions |
| nTree | a numeric value specifying number of trees to be constructed in model |

### Value

a vector or numeric predictions on user provided test data

### Examples

```
data = data1
trainX = as.matrix(data[c(1:100),2])
trainY = data[c(1:100),7]
testX = as.matrix(data[c(100:110),2])

Bart_prediction = BayesTreePCFit(trainX, trainY, testX)
```

## ComparePCurve *Power curve comparison*

### Description

Power curve comparison

## Usage

```
ComparePCurve(
  data,
  xCol,
  xCol.circ = NULL,
  yCol,
  testCol,
  testSet = NULL,
  thrs = 0.2,
  conflevel = 0.95,
  gridSize = c(50, 50),
  powerbins = 15,
  baseline = 1,
  limitMemory = TRUE,
  opt_method = "nlminb",
  sampleSize = list(optimSize = 500, bandSize = 5000),
  rngSeed = 1
)
```

## Arguments

| | |
|---|---|
| data | A list of data sets to be compared, the difference in the mean function is always computed as (f(data2) - f(data1)) |
| xCol | A numeric or vector stating column number of covariates |
| xCol.circ | A numeric or vector stating column number of circular covariates |
| yCol | A numeric value stating the column number of the response |
| testCol | A numeric/vector stating column number of covariates to used in generating test set. Maximum of two columns to be used. |
| testSet | A matrix or dataframe consisting of test points, default value NULL, if NULL computes test points internally using testCol variables. If not NULL, total number of test points must be less than or equal to 2500. |
| thrs | A numeric or vector representing threshold for each covariates |
| conflevel | A numeric between (0,1) representing the statistical significance level for constructing the band |
| gridSize | A numeric / vector to be used in constructing test set, should be provided when testSet is NuLL, else it is ignored. Default is `c(50,50)` for 2-dim input which is converted internally to a default of `c(1000)` for 1-dim input. Total number of test points (product of gridSize vector components) must be less than or equal to 2500. |
| powerbins | A numeric stating the number of power bins for computing the scaled difference, default is 15. |
| baseline | An integer between 0 to 2, where 1 indicates to use power curve of first dataset as the base for metric calculation, 2 indicates to use the power curve of second dataset as the base, and 0 indicates to use the average of both power curves as the base. Default is set to 1. |

| | |
|---|---|
| limitMemory | A boolean (True/False) indicating whether to limit the memory use or not. Default is true. If set to true, 5000 datapoints are randomly sampled from each dataset under comparison for inference |
| opt_method | A string specifying the optimization method to be used for hyperparameter estimation. Current options are: 'L-BFGS-B', 'BFGS', and 'nlminb'. Default is set to 'nlminb'. |
| sampleSize | A named list of two integer items: optimSize and bandSize, denoting the sample size for each dataset for hyperparameter optimization and confidence band computation, respectively, when limitMemory = TRUE. Default value is list(optimSize = 500,bandSize = 5000). |
| rngSeed | Random seed for sampling data when limitMemory = TRUE. Default is 1. |

## Value

a list containing :

- weightedDiff - a numeric, % difference between the functions weighted using the density of the covariates
- weightedStatDiff - a numeric, % statistically significant difference between the functions weighted using the density of the covariates
- scaledDiff - a numeric, % difference between the functions scaled to the orginal data
- scaledStatDiff - a numeric, % statistically significant difference between the functions scaled to the orginal data
- unweightedDiff - a numeric, % difference between the functions unweighted
- unweightedStatDiff - a numeric, % statistically significant difference between the functions unweighted
- reductionRatio - a list consisting of shrinkage ratio of features used in testSet
- mu1 - a vector of prediction on testset using the first data set
- mu2 - a vector of prediction on testset using the second data set
- muDiff - a vector of the difference in prediction (mu2 - mu1) for each test point
- band - a vector for the confidence band at all the testpoints for the two functions to be the same at a given cofidence level.
- confLevel - a numeric representing the statistical significance level for constructing the band
- testSet - a vector/matrix of the test points either provided by user, or generated internally
- estimatedParams - a list of estimated hyperaparameters for the Gaussian process model
- matchedData - a list of two matched datasets as generated by covariate matching

## References

For details, see Ding et al. (2020) available on arxiv at <https://arxiv.org/abs/2005.08652>.

## Examples

```
data1 = data1[1:100, ]
data2 = data2[1:100, ]
data = list(data1, data2)
xCol = 2
xCol.circ = NULL
yCol = 7
testCol = 2
testSet = NULL
thrs = 0.2
confLevel = 0.95
gridSize = 20
function_comparison = ComparePCurve(data, xCol, xCol.circ, yCol,
testCol, testSet, thrs, confLevel, gridSize)
```

---

# ComputeWeightedDifference

*Percentage weighted difference between power curves*

---

## Description

Computes percentage weighted difference between power curves based on user provided weights instead of the weights computed from the data. Please see `details` for more information.

## Usage

```
ComputeWeightedDifference(
  muDiff,
  weights,
  base,
  statDiff = FALSE,
  confBand = NULL
)
```

## Arguments

| | |
|---|---|
| `muDiff` | a vector of pointwise difference between two power curves on a testset as obtained from `ComparePCurve()` or `funGP()` function. |
| `weights` | a vector of user specified weights for each element of `muDiff`. It can be based on any probability distribution of user's choice. The weights must sum to 1. |
| `base` | a vector of predictions from a power curve; to be used as the denominator in computing the percentage difference. It can be either `mu1` or `mu2` as obtained from `ComparePCurve()` or `funGP()` function. |
| `statDiff` | a boolean specifying whether to compute the statistical significant difference or not. Default is set to `FALSE`, i.e. statistical significant difference is not computed. If set to `TRUE`, `confBand` must be provided. |

confBand — a vector of pointwise confidence band for all the points in the testset as obtained from ComparePCurve() or funGP() function, named as band. Should only be provided when statDiff is set to TRUE. Default value is NULL.

## Details

The function is a modification to the percentage weighted difference defined in Ding et. al. (2020). It computes a weighted difference between power curves on a testset, where the weights have to be provided by the user based on any probability distribution of their choice rather than the weights being computed from the data. The weights must sum to 1 to be valid.

## Value

a numeric percentage weighted difference or statistical significant percetage weighted difference based on whether statDiff is set to FALSE or TRUE.

## References

For details, see Ding et. al. (2020) available on arxiv at this link.

## Examples

```
ws_test = as.matrix(seq(4.5,8.5,length.out = 10))

userweights = dweibull(ws_test, shape = 2.25, scale = 6.5)
userweights = userweights/sum(userweights)
data1 = data1[1:100, ]
data2 = data2[1:100, ]
datalist = list(data1, data2)
xCol = 2
xCol.circ = NULL
yCol = 7
testCol = 2
output = ComparePCurve(data = datalist, xCol = xCol, yCol = yCol,
testCol = testCol, testSet = ws_test)
weightedDiff = ComputeWeightedDifference(output$muDiff, userweights, output$mu1)
weightedStatDiff = ComputeWeightedDifference(output$muDiff, userweights, output$mu1,
statDiff = TRUE, confBand = output$band)
```

---

# CovMatch *Covariate Matching*

---

## Description

The function aims to take list of two data sets and returns the after matched data sets using user specified covariates and threshold

## Usage

```
CovMatch(data, xCol, xCol.circ, thrs, priority)
```

## Arguments

| | |
|---|---|
| data | a list, consisting of data sets to match, also each of the individual data set can be dataframe or a matrix |
| xCol | a vector stating the column position of covariates used |
| xCol.circ | a vector stating the column position of circular variables |
| thrs | a numerical or a vector of threshold values for each covariates, against which matching happens It should be a single value or a vector of values representing threshold for each of the covariate |
| priority | a boolean, default value False, otherwise computes the sequence of matching |

## Value

a list containing :

- originalData - The data sets provided for matching
- matchedData - The data sets after matching
- MinMaxOriginal - The minimum and maximum value in original data for each covariate used in matching
- MinMaxMatched - The minimum and maximum value in matched data for each covariates used in matching

## References

Ding, Y. (2019). Data Science for Wind Energy. Chapman & Hall, Boca Raton, FL.

## Examples

```
data1 = data1[1:100, ]
data2 = data2[1:100, ]
data = list(data1, data2)
xCol = 2
xCol.circ = NULL
thrs = 0.1
priority = FALSE
matched_data = CovMatch(data, xCol, xCol.circ, thrs, priority)
```

## data1 *Wind Energy data set containing 47,542 data points*

### Description

A dataset containing the power produced and other attributes of almost 47,542 records.

### Usage

```
data(data1)
```

### Format

A data frame with 47,542 rows and 7 variables

### Details

- Data.point - sequence of integers displaying each record
- V - wind speed
- D - wind direction
- air.density - air density
- I - turbulence intensity
- S_b - wind shear
- Y - wind power

## data2 *Wind Energy data set containing 48,068 data points*

### Description

A dataset containing the power produced and other attributes of almost 48,068 records.

### Usage

```
data(data2)
```

### Format

A data frame with 48,068 rows and 7 variables

### Details

- Data.point - sequence of integers displaying each record
- V - wind speed
- D - wind direction
- air.density - air density
- I - turbulence intensity
- S_b - wind shear
- Y - wind power

---

`funGP` *Function comparison using Gaussian Process and Hypothesis testing*

---

### Description

Function comparison using Gaussian Process and Hypothesis testing

### Usage

```
funGP(
  datalist,
  xCol,
  yCol,
  confLevel = 0.95,
  testset,
  limitMemory = TRUE,
  opt_method = "nlminb",
  sampleSize = list(optimSize = 500, bandSize = 5000),
  rngSeed = 1
)
```

### Arguments

| | |
|---|---|
| `datalist` | A list of data sets to compute a function for each of them |
| `xCol` | A numeric or vector stating the column number of covariates |
| `yCol` | A numeric value stating the column number of target |
| `confLevel` | A single value representing the statistical significance level for constructing the band |
| `testset` | Test points at which the functions will be compared |
| `limitMemory` | A boolean (True/False) indicating whether to limit the memory use or not. Default is true. If set to true, 5000 datapoints are randomly sampled from each dataset under comparison for inference. |
| `opt_method` | A string specifying the optimization method to be used for hyperparameter estimation. Current options are: `'L-BFGS-B'`, `'BFGS'`, and `'nlminb'`. Default is set to `'nlminb'`. |

| | |
|---|---|
| sampleSize | A named list of two integer items: optimSize and bandSize, denoting the sample size for each dataset for hyperparameter optimization and confidence band computation, respectively, when limitMemory = TRUE. Default value is list(optimSize = 500,bandSize = 5000). |
| rngSeed | Random seed for sampling data when limitMemory = TRUE. Default is 1. |

## Value

a list containing :

- muDiff - A vector of pointwise difference between the predictions from the two datasets (mu2-mu1)
- mu1 - A vector of test prediction for first data set
- mu2 - A vector of test prediction for second data set
- band - A vector of the allowed statistical difference between functions at testpoints in testset
- confLevel - A numeric representing the statistical significance level for constructing the band
- testset - A matrix of test points to compare the functions
- estimatedParams - A list of estimated hyperparameters for GP

## References

Prakash, A., Tuo, R., & Ding, Y. (2020). "Gaussian process aided function comparison using noisy scattered data." arXiv preprint arXiv:2003.07899. <https://arxiv.org/abs/2003.07899>.

## Examples

```
datalist = list(data1[1:100,], data2[1:100, ])
xCol = 2
yCol = 7
confLevel = 0.95
testset = seq(4,10,length.out = 20)
function_diff = funGP(datalist, xCol, yCol, confLevel, testset)
```

---

# KnnPCFit *KNN : Fit*

---

## Description

The function models the powercurve using KNN, against supplied arguments

## Usage

```
KnnPCFit(data, xCol, yCol, subsetSelection = FALSE)
```

### Arguments

| | |
|---|---|
| data | a dataframe or a matrix, to be used in modelling |
| xCol | a vector or numeric values stating the column number of features |
| yCol | a numerical or a vector value stating the column number of target |
| subsetSelection | a boolean, default value is FALSE, if TRUE returns the best feature column number as xCol |

### Value

a list containing :

- data - The data set provided by user
- xCol - The column number of features provided by user or the best subset column number
- yCol - The column number of target provided by user
- bestK - The best k nearest neighbor calculated using the function
- RMSE - The RMSE calculated using the function for provided data using user defined features and best obtained K
- MAE - The MAE calculated using the function for provided data using user defined features and best obtained K

### Examples

```
data = data1[c(1:100),]
xCol = 2
yCol = 7
subsetSelection = FALSE

knn_model = KnnPCFit(data, xCol, yCol, subsetSelection)
```

---

## KnnPredict *KNN : Predict*

---

### Description

The function can be used to make prediction on test data using trained model

### Usage

```
KnnPredict(knnMdl, testData)
```

### Arguments

| | |
|---|---|
| knnMdl | a list containing:<br>• knnMdl$data - The data set provided by user<br>• knnMdl$xCol - The column number of features provided by user or the best subset column number<br>• knnMdl$yCol - The column number of target provided by user<br>• knn$bestK - The best k nearest neighbor calculated using the function KnnFit |
| testData | a data frame or matrix, to compute the predictions |

### Value

a numeric / vector with prediction on test data using model generated by KnnFit

### Examples

```
data = data1[c(1:100),]
xCol = 2
yCol = 7
subsetSelection = FALSE

knn_model = KnnPCFit(data, xCol, yCol, subsetSelection)
testData = data1[c(101:110), ]

prediction = KnnPredict(knn_model, testData)
```

---

## KnnUpdate *KNN : Update*

---

### Description

The function can be used to update KNN model when new data is provided

### Usage

```
KnnUpdate(knnMdl, newData)
```

### Arguments

| | |
|---|---|
| knnMdl | a list containing:<br>• knnMdl$data - The data set provided by user<br>• knnMdl$xCol - The column number of features provided by user or the best subset column number<br>• knnMdl$yCol - The column number of target provided by user |

- knn$bestK - The best k nearest neighbor calculated using the function KnnFit

`newData` a dataframe or a matrix, to be used for updating the model

## Value

a list containing :

- data - The updated data using old data set and new data
- xCol - The column number of features provided by user or the best subset column number
- yCol - The column number of target provided by user
- bestK - The best k nearest neighbor calculated for the new data using user specified features and target

## Examples

```
data = data1[c(1:100),]
xCol = 2
yCol = 7
subsetSelection = FALSE

knn_model = KnnPCFit(data, xCol, yCol, subsetSelection)
newData = data1[c(101:110), ]

knn_newmodel = KnnUpdate(knn_model, newData)
```

---

# predict.tempGP *predict from temporal Gaussian process*

---

## Description

predict function for tempGP objects. This function computes the prediction `f(x)` or `f(x) + g(t)` depending on the temporal distance between training and test points and whether the time indices for the test points are provided.

## Usage

```
## S3 method for class 'tempGP'
predict(object, testX, testT = NULL, trainT = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | An object of class tempGP. |
| testX | A matrix with each column corresponding to one input variable. |
| testT | A vector of time indices of the test points. When NULL, only function f(x) is used for prediction. Default is NULL. |
| trainT | Optional argument to override the existing trainT indices of the tempGP object. |
| ... | additional arguments for future development |

### Value

A vector of predictions at the testpoints in testX.

### Examples

```
data = DSWE::data1
trainindex = 1:100 #using the first 100 data points to train the model
traindata = data[trainindex,]
xCol = 2 #input variable columns
yCol = 7 #response column
trainX = as.matrix(traindata[,xCol])
trainY = as.numeric(traindata[,yCol])
tempGPObject = tempGP(trainX, trainY)
testdata = DSWE::data1[101:110,] # defining test data
testX = as.matrix(testdata[,xCol, drop = FALSE])
predF = predict(tempGPObject, testX)
```

---

## SplinePCFit *Smoothing spline Anova method*

---

### Description

Smoothing spline Anova method

### Usage

```
SplinePCFit(data, xCol, yCol, testX, modelFormula = NULL)
```

### Arguments

| | |
|---|---|
| data | a matrix or dataframe to be used in modelling |
| xCol | a numeric or vector stating the column number of feature covariates |
| yCol | a numeric value stating the column number of target |
| testX | a matrix or dataframe, to be used in computing the predictions |
| modelFormula | default is NULL else a model formula specifying target and features.Please refer 'gss' package documentation for more details |

## Value

a vector or numeric predictions on user provided test data

## Examples

```
data = data1[c(1:100),]
xCol = 2
yCol = 7
testX = data1[c(101:110), ]
Spline_prediction = SplinePCFit(data, xCol, yCol, testX)
```

---

`SvmPCFit` *SVM based power curve modelling*

---

## Description

SVM based power curve modelling

## Usage

```
SvmPCFit(trainX, trainY, testX, kernel = "radial")
```

## Arguments

| | |
|---|---|
| `trainX` | a matrix or dataframe to be used in modelling |
| `trainY` | a numeric or vector as a target |
| `testX` | a matrix or dataframe, to be used in computing the predictions |
| `kernel` | default is 'radial' else can be 'linear', 'polynomial' and 'sigmoid' |

## Value

a vector or numeric predictions on user provided test data

## Examples

```
data = data1
trainX = as.matrix(data[c(1:100),2])
trainY = data[c(1:100),7]
testX = as.matrix(data[c(101:110),2])

Svm_prediction = SvmPCFit(trainX, trainY, testX)
```

tempGP *temporal Gaussian process*

## Description

A Gaussian process based power curve model which explicitly models the temporal aspect of the power curve. The model consists of two parts: `f(x)` and `g(t)`.

## Usage

```
tempGP(trainX, trainY, trainT = NULL)
```

## Arguments

| | |
|---|---|
| `trainX` | A matrix with each column corresponding to one input variable. |
| `trainY` | A vector with each element corresponding to the output at the corresponding row of `trainX`. |
| `trainT` | A vector for time indices of the data points. By default, the function assigns natural numbers starting from 1 as the time indices. |

## Value

An object of class `tempGP` with the following attributes:

- trainX - same as the input matrix `trainX`.
- trainY - same as the input vector `trainY`.
- thinningNumber - the thinning number computed by the algorithm.
- modelF - A list containing the details of the model for predicting function `f(x)`:
  - X - The input variable matrix for computing the cross-covariance for predictions, same as `trainX` unless the model is updated. See `updateData.tempGP` method for details on updating the model.
  - y - The response vector, again same as `trainY` unless the model is updated.
  - weightedY - The weighted response, that is, the response left multiplied by the inverse of the covariance matrix.
- modelG - A list containing the details of the model for predicting function `g(t)`:
  - residuals - The residuals after subtracting function `f(x)` from the response. Used to predict `g(t)`. See `updateData.tempGP` method for updating the residuals.
  - time_index - The time indices of the residuals, same as `trainT`.
- estimatedParams - Estimated hyperparameters for function `f(x)`.
- llval - log-likelihood value of the hyperparameter optimization for `f(x)`.
- gradval - gradient vector at the optimal log-likelihood value.

## References

Prakash, A., Tuo, R., & Ding, Y. (2020). "The temporal overfitting problem with applications in wind power curve modeling." arXiv preprint arXiv:2012.01349. <https://arxiv.org/abs/2012.01349>.

## See Also

predict.tempGP for computing predictions and updateData.tempGP for updating data in a tempGP object.

## Examples

```
data = DSWE::data1
trainindex = 1:100 #using the first 100 data points to train the model
traindata = data[trainindex,]
xCol = 2 #input variable columns
yCol = 7 #response column
trainX = as.matrix(traindata[,xCol])
trainY = as.numeric(traindata[,yCol])
tempGPObject = tempGP(trainX, trainY)
```

---

# updateData *Updating data in a model*

---

## Description

updateData is a generic function to update data in a model.

## Usage

```
updateData(object, ...)
```

## Arguments

| | |
|---|---|
| object | A model object |
| ... | additional arguments for passing to specific methods |

## Value

The returned value would depend on the class of its argument object.

## See Also

updateData.tempGP

`updateData.tempGP` *Update the data in a tempGP object*

## Description

This function updates `trainX`, `trainY`, and `trainT` in a `tempGP` object. By default, if the new data has `m` data points, the function removes top `m` data points from the tempGP object and appends the new data at the bottom, thus keeping the total number of data points the same. This can be overwritten by setting `replace = FALSE` to keep all the data points (old and new). The method also updates `modelG` by computing and updating residuals at the new data points. `modelF` can be also be updated by setting the argument `updateModelF` to `TRUE`, though not required generally (see comments in the `Arguments`.)

## Usage

```
## S3 method for class 'tempGP'
updateData(
  object,
  newX,
  newY,
  newT = NULL,
  replace = TRUE,
  updateModelF = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | An object of class tempGP. |
| `newX` | A matrix with each column corresponding to one input variable. |
| `newY` | A vector with each element corresponding to the output at the corresponding row of `newX`. |
| `newT` | A vector with time indices of the new datapoints. If `NULL`, the function assigns natural numbers starting with one larger than the existing time indices in `trainT`. |
| `replace` | A boolean to specify whether to replace the old data with the new one, or to add the new data while still keeping all the old data. Default is TRUE, which replaces the top `m` rows from the old data, where `m` is the number of data points in the new data. |
| `updateModelF` | A boolean to specify whether to update `modelF` as well. If the original `tempGP` model is trained on a sufficiently large dataset (say one year), updating `modelF` regularly may not result in any significant improvement, but can be computationally expensive. |
| `...` | additional arguments for future development |

## Value

An updated object of class tempGP.

## Examples

```
data = DSWE::data1
trainindex = 1:100 #using the first 100 data points to train the model
traindata = data[trainindex,]
xCol = 2 #input variable columns
yCol = 7 #response column
trainX = as.matrix(traindata[,xCol])
trainY = as.numeric(traindata[,yCol])
tempGPObject = tempGP(trainX, trainY)
newdata = DSWE::data1[101:110,] # defining new data
newX = as.matrix(newdata[,xCol, drop = FALSE])
newY = as.numeric(newdata[,yCol])
tempGPupdated = updateData(tempGPObject, newX, newY)
```

# Index