In [ ]:
```
using LinearAlgebra
using Statistics
```

In [ ]:
```
"""
GEKPLS(x, y, x_matrix, y_matrix, grads, xlimits, delta_x, extra_points, n_comp, bet
                  reduced_likelihood_function_value,
                  X_offset, X_scale, X_after_std, pls_mean_reshaped, y_mean, y_std)
"""


abstract type AbstractSurrogate end
mutable struct GEKPLS{T, X, Y} <: AbstractSurrogate
    x::X
    y::Y
    x_matrix::Matrix{T} #1
    y_matrix::Matrix{T} #2
    grads::Matrix{T} #3
    xl::Matrix{T} #xlimits #4
    delta::T #5
    extra_points::Int #6
    num_components::Int #7
    beta::Vector{T} #8
    gamma::Matrix{T} #9
    theta::Vector{T} #10
    reduced_likelihood_function_value::T #11
    X_offset::Matrix{T} #12
    X_scale::Matrix{T} #13
    X_after_std::Matrix{T} #14 - X after standardization
    pls_mean::Matrix{T} #15
    y_mean::T #16
    y_std::T #17
end

"""
    bounds_error(x, xl)
Checks if input x values are within range of upper and lower bounds
"""
function bounds_error(x, xl)
    num_x_rows = size(x, 1)
    num_dim = size(xl, 1)
    for i in 1:num_x_rows
        for j in 1:num_dim
            if (x[i, j] < xl[j, 1] || x[i, j] > xl[j, 2])
                return true
            end
        end
    end
    return false
end

"""
    GEKPLS(X, y, grads, n_comp, delta_x, lb, ub, extra_points, theta)

Constructor for GEKPLS Struct
```

```
- x_vec: vector of tuples with x values
- y_vec: vector of floats with outputs
- grads_vec: gradients associated with each of the X points
- n_comp: number of components
- lb: lower bounds
- ub: upper bounds
- delta_x: step size while doing Taylor approximation
- extra_points: number of points to consider
- theta: initial expected variance of PLS regression components
"""



function GEKPLS(x_vec, y_vec, grads_vec, n_comp, delta_x, lb, ub, extra_points, the
    xlimits = hcat(lb, ub)
    X = vector_of_tuples_to_matrix(x_vec)
    y = reshape(y_vec, (size(X, 1), 1))
    grads = vector_of_tuples_to_matrix2(grads_vec)

    #ensure that X values are within the upper and lower bounds
    if bounds_error(X, xlimits)
        println("X values outside bounds")
        return
    end

    pls_mean, X_after_PLS, y_after_PLS = _ge_compute_pls(X, y, n_comp, grads, delta
        xlimits, extra_points)
    X_after_std, y_after_std, X_offset, y_mean, X_scale, y_std = standardization(X_
        y_after_PLS)
    D, ij = cross_distances(X_after_std)
    pls_mean_reshaped = reshape(pls_mean, (size(X, 2), n_comp))
    d = componentwise_distance_PLS(D, "squar_exp", n_comp, pls_mean_reshaped)
    nt, nd = size(X_after_PLS)
    beta, gamma, reduced_likelihood_function_value = _reduced_likelihood_function(t
        "squar_exp",
        d, nt, ij,
        y_after_std)
    return GEKPLS(x_vec, y_vec, X, y, grads, xlimits, delta_x, extra_points, n_comp
        gamma, theta,
        reduced_likelihood_function_value,
        X_offset, X_scale, X_after_std, pls_mean_reshaped, y_mean, y_std)
end

"""
    (g::GEKPLS)(X_test)

    Take in a set of one or more points and provide predicted approximate outputs (
"""
function (g::GEKPLS)(x_vec)
    _check_dimension(g, x_vec)
    X_test = prep_data_for_pred(x_vec)
    n_eval, n_features_x = size(X_test)
    X_cont = (X_test .- g.X_offset) ./ g.X_scale
    dx = differences(X_cont, g.X_after_std)
    pred_d = componentwise_distance_PLS(dx, "squar_exp", g.num_components, g.pls_me
    nt = size(g.X_after_std, 1)
    r = transpose(reshape(squar_exp(g.theta, pred_d), (nt, n_eval)))
    f = ones(n_eval, 1)
    y_ = (f * g.beta) + (r * g.gamma)
    y = g.y_mean .+ g.y_std * y_
    return y[1]
end

"""
    add_point!(g::GEKPLS, new_x, new_y, new_grads)

add a new point to the dataset.
"""
function add_point!(g::GEKPLS, x_tup, y_val, grad_tup)
    new_x = prep_data_for_pred(x_tup)
    new_grads = prep_data_for_pred(grad_tup)
    if vec(new_x) in eachrow(g.x_matrix)
        println("Adding a sample that already exists. Cannot build GEKPLS")
        return
    end

    if bounds_error(new_x, g.xl)
        println("x values outside bounds")
        return
    end
    temp_y = copy(g.y) #without the copy here, we get error ("cannot resize array w
    push!(g.x, x_tup)
    push!(temp_y, y_val)
    g.y = temp_y
    g.x_matrix = vcat(g.x_matrix, new_x)
    g.y_matrix = vcat(g.y_matrix, y_val)
    g.grads = vcat(g.grads, new_grads)
    pls_mean, X_after_PLS, y_after_PLS = _ge_compute_pls(g.x_matrix, g.y_matrix,
        g.num_components,
        g.grads, g.delta, g.xl,
        g.extra_points)
    g.X_after_std, y_after_std, g.X_offset, g.y_mean, g.X_scale, g.y_std = standard
        y_after_PLS)
    D, ij = cross_distances(g.X_after_std)
    g.pls_mean = reshape(pls_mean, (size(g.x_matrix, 2), g.num_components))
    d = componentwise_distance_PLS(D, "squar_exp", g.num_components, g.pls_mean)
    nt, nd = size(X_after_PLS)
    g.beta, g.gamma, g.reduced_likelihood_function_value = _reduced_likelihood_func
        "squar_exp",
        d,
        nt,
        ij,
        y_after_std)
end

"""
    _ge_compute_pls(X, y, n_comp, grads, delta_x, xlimits, extra_points)
Gradient-enhanced PLS-coefficients.
Parameters
----------
- X: [n_obs,dim] - The input variables.
- y: [n_obs,ny] - The output variable
- n_comp: int - Number of principal components used.
- gradients: - The gradient values. Matrix size (n_obs,dim)
- delta_x: real - The step used in the First Order Taylor Approximation
- xlimits: [dim, 2]- The upper and lower var bounds.
- extra_points: int - The number of extra points per each training point.
Returns
-------
- Coeff_pls: [dim, n_comp] - The PLS-coefficients.
- X: Concatenation of XX: [extra_points*nt, dim] - Extra points added (when extra_p
- y: Concatenation of yy[extra_points*nt, 1]- Extra points added (when extra_points
"""
function _ge_compute_pls(X, y, n_comp, grads, delta_x, xlimits, extra_points)

    # this function is equivalent to a combination of
    # https://github.com/SMTorg/smt/blob/f124c01ffa78c04b80221dded278a20123dac742/s
    # and https://github.com/SMTorg/smt/blob/f124c01ffa78c04b80221dded278a20123dac7

    nt, dim = size(X)
    XX = zeros(0, dim)
    yy = zeros(0, size(y)[2])
    coeff_pls = zeros((dim, n_comp, nt))

    for i in 1:nt
        if dim >= 3
            bb_vals = circshift(boxbehnken(dim, 1), 1)
        else
            bb_vals = [0.0 0.0; #center
                1.0 0.0; #right
                0.0 1.0; #up
                -1.0 0.0; #left
                0.0 -1.0; #down
                1.0 1.0; #right up
                -1.0 1.0; #left up
                -1.0 -1.0; #left down
                1.0 -1.0]
        end
        _X = zeros((size(bb_vals)[1], dim))
        _y = zeros((size(bb_vals)[1], 1))
        bb_vals = bb_vals .* (delta_x * (xlimits[:, 2] - xlimits[:, 1]))' #smt call
        _X = X[i, :]' .+ bb_vals
        bb_vals = bb_vals .* grads[i, :]'
        _y = y[i, :] .+ sum(bb_vals, dims = 2)

        #_pls.fit(_X, _y) # relic from sklearn version; retained for future referen
        #coeff_pls[:, :, i] = _pls.x_rotations_ #relic from sklearn version; retain

        coeff_pls[:, :, i] = _modified_pls(_X, _y, n_comp) #_modified_pls returns t
        if extra_points != 0
            start_index = max(1, length(coeff_pls[:, 1, i]) - extra_points + 1)
            max_coeff = sortperm(broadcast(abs, coeff_pls[:, 1, i]))[start_index:en
            for ii in max_coeff
                XX = [XX; transpose(X[i, :])]
                XX[end, ii] += delta_x * (xlimits[ii, 2] - xlimits[ii, 1])
                yy = [yy; y[i]]
                yy[end] += grads[i, ii] * delta_x * (xlimits[ii, 2] - xlimits[ii, 1
            end
        end
    end
    if extra_points != 0
        X = [X; XX]
        y = [y; yy]
    end

    pls_mean = mean(broadcast(abs, coeff_pls), dims = 3)
    return pls_mean, X, y
end

######start of bbdesign######

#
# Adapted from 'ExperimentalDesign.jl: Design of Experiments in Julia'
# https://github.com/phrb/ExperimentalDesign.jl

# MIT License

# ExperimentalDesign.jl: Design of Experiments in Julia
# Copyright (C) 2019 Pedro Bruel <pedro.bruel@gmail.com>

# Permission is hereby granted, free of charge,  to any person obtaining a copy of
# this software  and associated documentation  files (the "Software"), to  deal in
# the Software  without restriction,  including without  limitation the  rights to
# use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
# the Software, and to permit persons to  whom the Software is furnished to do so,
# subject to the following conditions:

# The  above copyright  notice  and  this permission  notice  (including the  next
# paragraph)  shall be  included  in all  copies or  substantial  portions of  the
# Software.

# THE  SOFTWARE IS  PROVIDED "AS  IS", WITHOUT  WARRANTY OF  ANY KIND,  EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
# FOR A PARTICULAR  PURPOSE AND NONINFRINGEMENT. IN NO EVENT  SHALL THE AUTHORS OR
# COPYRIGHT HOLDERS BE  LIABLE FOR ANY CLAIM, DAMAGES OR  OTHER LIABILITY, WHETHER
# IN  AN ACTION  OF  CONTRACT, TORT  OR  OTHERWISE,  ARISING FROM,  OUT  OF OR  IN
# CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#

function boxbehnken(matrix_size::Int)
    boxbehnken(matrix_size, matrix_size)
end

function boxbehnken(matrix_size::Int, center::Int)
    @assert matrix_size >= 3

    A_fact = explicit_fullfactorial(Tuple([-1, 1] for i in 1:2))

    rows = floor(Int, (0.5 * matrix_size * (matrix_size - 1)) * size(A_fact)[1])

    A = zeros(rows, matrix_size)

    l = 0
    for i in 1:(matrix_size - 1)
```

```
        for j in (i + 1):matrix_size
            l = l + 1
            A[(max(0, (l - 1) * size(A_fact)[1]) + 1):(l * size(A_fact)[1]), i] = A
                1]
            A[(max(0, (l - 1) * size(A_fact)[1]) + 1):(l * size(A_fact)[1]), j] = A
                2]
        end
    end

    if center == matrix_size
        if matrix_size <= 16
            points = [0, 0, 3, 3, 6, 6, 6, 8, 9, 10, 12, 12, 13, 14, 15, 16]
            center = points[matrix_size]
        end
    end

    A = transpose(hcat(transpose(A), transpose(zeros(center, matrix_size))))
end

function explicit_fullfactorial(factors::Tuple)
    explicit_fullfactorial(fullfactorial(factors))
end

function explicit_fullfactorial(iterator::Base.Iterators.ProductIterator)
    hcat(vcat.(collect(iterator)...)...)
end

function fullfactorial(factors::Tuple)
    Base.Iterators.product(factors...)
end

######end of bb design######

"""
We subtract the mean from each variable. Then, we divide the values of each
variable by its standard deviation.

Parameters
----------

X - The input variables.
y - The output variable.

Returns
-------

X: [n_obs, dim]
   The standardized input matrix.

y: [n_obs, 1]
   The standardized output vector.

X_offset: The mean (or the min if scale_X_to_unit=True) of each input variable.

y_mean: The mean of the output variable.

```

```
X_scale:  The standard deviation of each input variable.

y_std: The standard deviation of the output variable.

"""
function standardization(X, y)
    #Equivalent of https://github.com/SMTorg/smt/blob/4a4df255b9259965439120091007f
    X_offset = mean(X, dims = 1)
    X_scale = std(X, dims = 1)
    X_scale = map(x -> (x == 0.0) ? x = 1 : x = x, X_scale) #to prevent division by
    y_mean = mean(y)
    y_std = std(y)
    y_std = map(y -> (y == 0) ? y = 1 : y = y, y_std) #to prevent division by 0 bel
    X = (X .- X_offset) ./ X_scale
    y = (y .- y_mean) ./ y_std
    return X, y, X_offset, y_mean, X_scale, y_std
end

"""
Computes the nonzero componentwise cross-distances between the vectors
in X

Parameters
----------

X: [n_obs, dim]

Returns
-------
D:  [n_obs * (n_obs - 1) / 2, dim]
    - The cross-distances between the vectors in X.

ij: [n_obs * (n_obs - 1) / 2, 2]
        - The indices i and j of the vectors in X associated to the cross-
          distances in D.
"""
function cross_distances(X)
    # equivalent of https://github.com/SMTorg/smt/blob/4a4df255b9259965439120091007
    n_samples, n_features = size(X)
    n_nonzero_cross_dist = (n_samples * (n_samples - 1)) ÷ 2
    ij = zeros((n_nonzero_cross_dist, 2))
    D = zeros((n_nonzero_cross_dist, n_features))
    ll_1 = 0

    for k in 1:(n_samples - 1)
        ll_0 = ll_1 + 1
        ll_1 = ll_0 + n_samples - k - 1
        ij[ll_0:ll_1, 1] .= k
        ij[ll_0:ll_1, 2] = (k + 1):1:n_samples
        D[ll_0:ll_1, :] = -(X[(k + 1):n_samples, :] .- X[k, :]')
    end
    return D, Int.(ij)
end

"""
        Computes the nonzero componentwise cross-spatial-correlation-distance
        between the vectors in X.

        Equivalent of https://github.com/SMTorg/smt/blob/4a4df255b92599654391200910
        with some simplifications (removed theta and return_derivative as it's not

        Parameters
        ----------

        D: [n_obs * (n_obs - 1) / 2, dim]
            - The L1 cross-distances between the vectors in X.

        corr: str
                - Name of the correlation function used.
                squar_exp or abs_exp.

        n_comp: int
                - Number of principal components used.

        coeff_pls: [dim, n_comp]
                - The PLS-coefficients.

        Returns
        -------

        D_corr: [n_obs * (n_obs - 1) / 2, n_comp]
                - The componentwise cross-spatial-correlation-distance between the
                vectors in X.

"""
function componentwise_distance_PLS(D, corr, n_comp, coeff_pls)

    #equivalent of https://github.com/SMTorg/smt/blob/4a4df255b92599654391200091007f
    #todo
    #figure out how to handle this computation in the case of very large matrices
    #similar to what SMT has done
    #at https://github.com/SMTorg/smt/blob/4a4df255b925996543912009l007f9852f41523e
    D_corr = zeros((size(D)[1], n_comp))

    if corr == "squar_exp"
        D_corr = D .^ 2 * coeff_pls .^ 2
    else #abs_exp
        D_corr = abs.(D) * abs.(coeff_pls)
    end

    return D_corr
end

"""
Squared exponential correlation model.
Equivalent of https://github.com/SMTorg/smt/blob/4a4df255b92599654391200091007f9852f
Parameters:
-----------
theta : Hyperparameters of the correlation model
d: componentwise distances from componentwise_distance_PLS

Returns:
```

```
--------
r:  array containing the values of the autocorrelation model

"""
function squar_exp(theta, d)
    n_components = size(d)[2]
    theta = reshape(theta, (1, n_components))
    return exp.(-sum(theta .* d, dims = 2))
end

"""
    differences(X, Y)
return differences between two arrays

given an input like this:

X = [1.0 1.0 1.0; 2.0 2.0 2.0]
Y = [1.0 2.0 3.0; 4.0 5.0 6.0; 7.0 8.0 9.0]
diff = differences(X,Y)

We get an output (diff) that looks like this:

[ 0. -1. -2.
 -3. -4. -5.
 -6. -7. -8.
  1.  0. -1.
 -2. -3. -4.
 -5. -6. -7.]

"""
function differences(X, Y)
    #equivalent of https://github.com/SMTorg/smt/blob/4a4df255b9259965439120091007f
    #code credit: Elias Carvalho - https://stackoverflow.com/questions/72392010/row
    Rx = repeat(X, inner = (size(Y, 1), 1))
    Ry = repeat(Y, size(X, 1))
    return Rx - Ry
end

"""
    _reduced_likelihood_function(theta, kernel_type, d, nt, ij, y_norma, noise = 0.

Compute the reduced likelihood function value and other coefficients necessary for
This function is a loose translation of SMT code from
https://github.com/SMTorg/smt/blob/4a4df255b9259965439120091007f9852f41523e/smt/sur
It  determines the BLUP parameters and evaluates the reduced likelihood function fo

Parameters
----------
theta: array containing the parameters at which the Gaussian Process model paramete
kernel_type: name of the correlation function.
d: The componentwise cross-spatial-correlation-distance between the vectors in X.
nt: number of training points
ij: The indices i and j of the vectors in X associated to the cross-distances in D.
y_norma: Standardized y values
noise: noise hyperparameter - increasing noise reduces reduced_likelihood_function_
```

```
Returns
-------
reduced_likelihood_function_value: real
    - The value of the reduced likelihood function associated with the given autoco
beta:  Generalized least-squares regression weights
gamma: Gaussian Process weights.

"""
function _reduced_likelihood_function(theta, kernel_type, d, nt, ij, y_norma, noise
    #equivalent of https://github.com/SMTorg/smt/blob/4a4df255b9259965439120091007f
    reduced_likelihood_function_value = -Inf
    nugget = 1000000.0 * eps() #a jitter for numerical stability; reducing the mult
    if kernel_type == "squar_exp" #todo - add other kernel type abs_exp etc.
        r = squar_exp(theta, d)
    end
    R = (I + zeros(nt, nt)) .* (1.0 + nugget + noise)

    for k in 1:size(ij)[1]
        R[ij[k, 1], ij[k, 2]] = r[k]
        R[ij[k, 2], ij[k, 1]] = r[k]
    end

    C = cholesky(R).L #todo - #values diverge at this point from SMT code; verify i
    F = ones(nt, 1) #todo - examine if this should be a parameter for this function
    Ft = C \ F
    Q, G = qr(Ft)
    Q = Array(Q)
    Yt = C \ y_norma
    #todo - in smt, they check if the matrix is ill-conditioned using SVD. Verify a
    beta = G \ [(transpose(Q) ⋅ Yt)]
    rho = Yt .- (Ft .* beta)
    gamma = transpose(C) \ rho
    sigma2 = sum((rho) .^ 2, dims = 1) / nt
    detR = prod(diag(C) .^ (2.0 / nt))
    reduced_likelihood_function_value = -nt * log10(sum(sigma2)) - nt * log10(detR)
    return beta, gamma, reduced_likelihood_function_value
end

### MODIFIED PLS BELOW ###

# The code below is a simplified version of
# SKLearn's PLS
# https://github.com/scikit-learn/scikit-learn/blob/80598905e/sklearn/cross_decompo
# It is completely self-contained (no dependencies)

function _center_scale(X, Y)
    x_mean = mean(X, dims = 1)
    X .-= x_mean
    y_mean = mean(Y, dims = 1)
    Y .-= y_mean
    x_std = std(X, dims = 1)
    x_std[x_std .== 0] .= 1.0
    X ./= x_std
    y_std = std(Y, dims = 1)
    y_std[y_std .== 0] .= 1.0
    Y ./= y_std
    return X, Y
end

function _svd_flip_1d(u, v)
    # equivalent of https://github.com/scikit-learn/scikit-learn/blob/80598905e5177
    biggest_abs_val_idx = findmax(abs.(vec(u)))[2]
    sign_ = sign(u[biggest_abs_val_idx])
    u .*= sign_
    v .*= sign_
end

function _get_first_singular_vectors_power_method(X, Y)
    my_eps = eps()
    y_score = vec(Y)
    x_weights = transpose(X)y_score / dot(y_score, y_score)
    x_weights ./= (sqrt(dot(x_weights, x_weights)) + my_eps)
    x_score = X * x_weights
    y_weights = transpose(Y)x_score / dot(x_score, x_score)
    y_score = Y * y_weights / (dot(y_weights, y_weights) + my_eps)
    #Equivalent in intent to https://github.com/scikit-learn/scikit-learn/blob/8059
    if any(isnan.(x_weights)) || any(isnan.(y_weights))
        return false, false
    end
    return x_weights, y_weights
end

function _modified_pls(X, Y, n_components)
    x_weights_ = zeros(size(X, 2), n_components)
    _x_scores = zeros(size(X, 1), n_components)
    x_loadings_ = zeros(size(X, 2), n_components)
    Xk, Yk = _center_scale(X, Y)

    for k in 1:n_components
        x_weights, y_weights = _get_first_singular_vectors_power_method(Xk, Yk)

        if x_weights == false
            break
        end

        _svd_flip_1d(x_weights, y_weights)
        x_scores = Xk * x_weights
        x_loadings = transpose(x_scores)Xk / dot(x_scores, x_scores)
        Xk = Xk - (x_scores * x_loadings)
        y_loadings = transpose(x_scores) * Yk / dot(x_scores, x_scores)
        Yk = Yk - x_scores * y_loadings
        x_weights_[:, k] = x_weights
        _x_scores[:, k] = x_scores
        x_loadings_[:, k] = vec(x_loadings)
    end

    x_rotations_ = x_weights_ * pinv(transpose(x_loadings_)x_weights_)
    return x_rotations_
end

### MODIFIED PLS ABOVE ###
```

```julia
### BELOW ARE HELPER FUNCTIONS TO HELP MODIFY VECTORS INTO ARRAYS

function vector_of_tuples_to_matrix(v)
    num_rows = length(v)
    num_cols = length(first(v))
    K = zeros(num_rows, num_cols)
    for row in 1:num_rows
        for col in 1:num_cols
            K[row, col] = v[row][col]
        end
    end
    return K
end

function vector_of_tuples_to_matrix2(v)
    #convert gradients into matrix form
    num_rows = length(v)
    num_cols = length(first(first(v)))
    K = zeros(num_rows, num_cols)
    for row in 1:num_rows
        for col in 1:num_cols
            K[row, col] = v[row][1][col]
        end
    end
    return K
end

function prep_data_for_pred(v)
    l = length(first(v))
    if (l == 1)
        return [tup[k] for k in 1:1, tup in v]
    end
    return [tup[k] for tup in v, k in 1:l]
end
```

```
prep_data_for_pred (generic function with 1 method)
```

In [ ]:

```julia
x_vec = [(1.0, 2.0), (3.0, 4.0), (5.0, 6.0)]  # Example x values as tuples
y_vec = [10.0, 20.0, 30.0]  # Example y values
grads_vec = [((1.0, 1.0),), ((2.0, 2.0),), ((3.0, 3.0),)]  # Example gradients as t
n_comp = 2  # Number of components
delta_x = 0.1  # Example delta_x value
lb = [0.0, 0.0]  # Lower bounds
ub = [10.0, 10.0]  # Upper bounds
extra_points = 5  # Example number of extra points
theta = [0.5, 0.5]  # Example theta values
```

```
2-element Vector{Float64}:
 0.5
 0.5
```

In [ ]:

```julia
using ProfileView
using Profile

@profile GEKPLS(x_vec, y_vec, grads_vec, n_comp, delta_x, lb, ub, extra_points, the
ProfileView.view()
```

```
# @profview GEKPLS(x_vec, y_vec, grads_vec, n_comp, delta_x, lb, ub, extra_points,
```

```
Gtk.GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible=TR
UE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALSE, is-focu
s=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALSE, receives-defau
lt=FALSE, composite-child=FALSE, style, events=0, no-show-all=FALSE, has-tooltip=F
ALSE, tooltip-markup=NULL, tooltip-text=NULL, window, opacity=1.000000, double-buf
fered, halign=GTK_ALIGN_FILL, valign=GTK_ALIGN_FILL, margin-left, margin-right, ma
rgin-start=0, margin-end=0, margin-top=0, margin-bottom=0, margin=0, hexpand=FALS
E, vexpand=FALSE, hexpand-set=FALSE, vexpand-set=FALSE, expand=FALSE, scale-factor
=1, border-width=0, resize-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Profile",
role=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-
width=800, default-height=600, destroy-with-parent=FALSE, hide-titlebar-when-maxim
ized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NORMAL, s
kip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, accept-focus=TR
UE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity=GDK_GRAVITY_NORTH_W
EST, transient-for, attached-to, has-resize-grip, resize-grip-visible, applicatio
n, is-active=TRUE, has-toplevel-focus=TRUE, startup-id, mnemonics-visible=FALSE, f
ocus-visible=FALSE, is-maximized=FALSE)
```

```
In [ ]:
```