# Point Registration with Gaussian RBF

October 12, 2021

## S4.2 Linear least-squares estimation

This problem is to estimate transformation relationship between source (control) points and target points by solving a least-squares problem. And then find transformed query points using the estimated solution. This algorithm can be applied on image registration by selecting some landmarks on image as control points. Steps to finish this task is:

1. **Fit Stage:**
   (a) Compute $\mathbf{K}$: $K_{ij} = R(\|\mathbf{c}_i - \mathbf{c}_j\|)$
   (b) Solve transformation relationship using $\mathbf{K}$: $\hat{\boldsymbol{\alpha}} = arg \min_{\boldsymbol{\alpha}} \|(\mathbf{K} + \lambda \mathbf{I})\boldsymbol{\alpha} - \mathbf{t}\|$
2. **Evaluate Stage:**
   (a) Compute $\mathbf{K}$: $K_{ij} = R(\|\mathbf{q}_i - \mathbf{c}_j\|)$
   (b) Estimate transformed query points: $\mathbf{K}\boldsymbol{\alpha}$

where $\mathbf{c}$, $\mathbf{t}$, and $\mathbf{q}$ represent source (control) points, target (transformed source) points, and query points respectively. $R(r)$ denotes a Gaussian kernel, i.e., $R(r) = e^{-r^2/2\sigma^2}$.

Write a function that implements above algorithm to estimate transformed query points on 2D level.

**Solution:**

```julia
using LinearAlgebra
using SparseArrays
using Plots

"""QT = registration(C, T, Q, Lambda, Sigma)
Project each column of `X` onto the orthogonal complement of the null space
of the input matrix `A`.
In: * `C`: control points, a `M  2` matrix
* `T`: target points, a `M  2` matrix
* `Q`: query points, a `N  2` matrix
* `Lambda`: approximation parameter
* `Sigma`: kernel radius
Out: * `QT` : transformed query points, a `N  2` matrix"""
function registration(C, T, Q, Lambda, Sigma)
    M = size(C,1)
    N = size(Q,1)
    R = r -> exp(-r^2/(2*Sigma^2))
    K = [R(norm(C[i,:]-C[j,:])) for i=1:M, j=1:M]
    F = svd(K+Lambda*spdiagm(0=>ones(M)))
    S = [1/F.S[i] for i=1:M]
    alpha = F.V*spdiagm(0=>S)*F.U'*T
    K = [R(norm(Q[i,:]-C[j,:])) for i=1:N, j=1:M]
    return K*alpha
end

# test
C = [1 0; 0 1; -1 0; 0 -1;]
T = [1 0.5; 0 2.5; -1 0.5; 0 -1.5;]+rand(4,2)*0.1
Q = hcat([cos(pi/20*i) for i=1:40], [sin(pi/20*i) for i=1:40])
QT = registration(C, T, Q, 0.01, 1)
scatter(Q[:,1], Q[:,2], label="Q", xlim=(-2,2))
scatter!(QT[:,1], QT[:,2], label="QT")
scatter!(C[:,1], C[:,2], label="C")
scatter!(T[:,1], T[:,2], label="T")
```

what is X and A?
confusing docstring here!

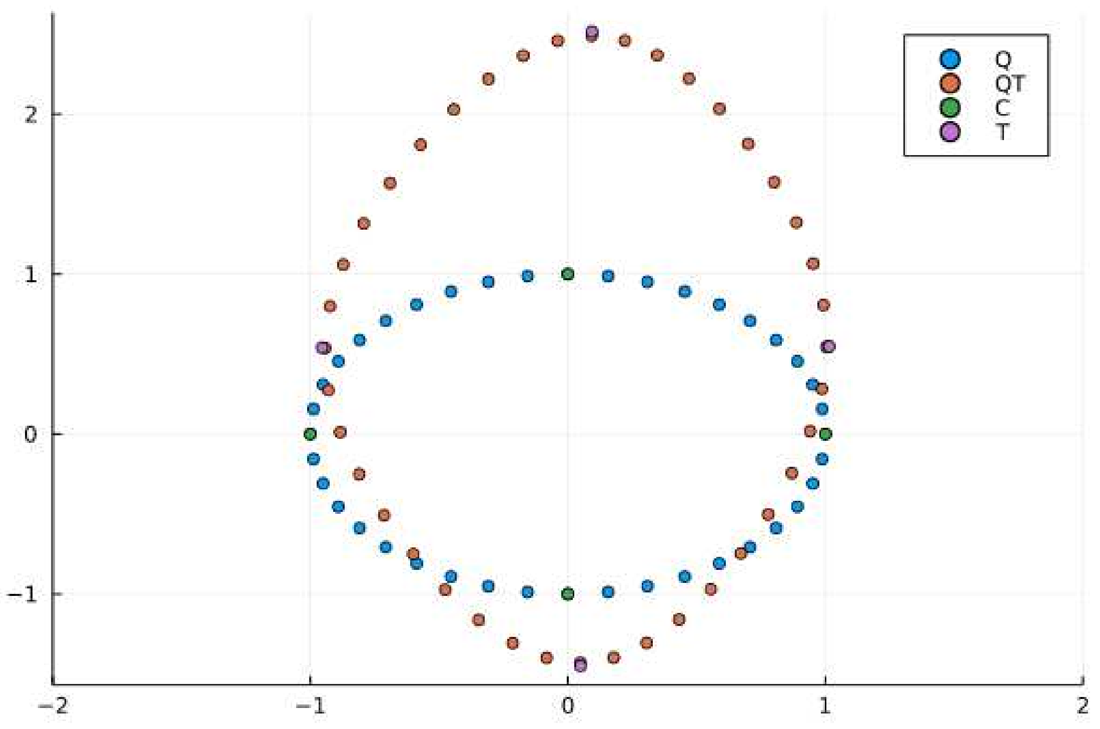

Figure 1: Test Visualisation

@538 @632 S4.2

## S4.2 The pseudo-inverse of a matrix.

## Problem description:

Please implement the function of the pseudoinverse of a matrix (don’t use Julia function pinv). Please consider the situation when the matrix is a zero matrix. (hint: use svd() , compare the result with pinv)

## Function Specification:

""""

R = PseudoInverse(A)

Calculate pseudo-inverse of any matrix A

In:

`A` `M × N` matrix

Out:

'A_PI' pseudo-inverse of matrix A `N × M` matrix

""""

function PseudoInverse(A)

## Code Answer:

```julia
"""

R = PseudoInverse(A)

Calculate pseudo-inverse of any matrix A

In:


`A` `M × N` matrix

Out:

'A_PI' pseudo-inverse of matrix A `N × M` matrix


"""

function PseudoInverse(A)
    U,s,V = svd(A)

    r = rank(A)

    if r > 0
        Vr = V[:,1:r]
        Ur = U[:,1:r]
        s = s[1:r]
        Sr = Diagonal(s)
    end

    if r==0
        return A'
    end

    A_PI = Vr * inv(Sr) * Ur'

    return A_PI
end
```

this is inefficient.
use rank(Diagonal(s))

**S4.1** (Regularized multivariate linear least squares)

Assume that we are given data $(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, \ldots, N$ consisting of pairs of features $\boldsymbol{x}_n \in \mathbb{R}^M$ and responses $\boldsymbol{y}_n \in \mathbb{R}^P$. Our task is to find a linear map that best translates any given feature to the corresponding response. Such a linear map can be represented by a matrix $\boldsymbol{W} \in \mathbb{R}^{M \times P}$ whose predicted response $\widehat{\boldsymbol{y}}_n$ is given by $\widehat{\boldsymbol{y}}_n = \boldsymbol{W}'\boldsymbol{x}_n$.

Specifically, we want to find a matrix $\widehat{\boldsymbol{W}}$ that minimizes the loss $L_\beta$ given below:

$$\widehat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} L_\beta(\boldsymbol{W}), \quad L_\beta(\boldsymbol{W}) = \sum_{n=1}^{N} \ell(\boldsymbol{y}_n, \widehat{\boldsymbol{y}}_n) + \beta \|\boldsymbol{W}\|_F^2$$

where $\ell(\boldsymbol{y}_n, \widehat{\boldsymbol{y}}_n) = \frac{1}{2} \|\boldsymbol{y}_n - \widehat{\boldsymbol{y}}_n\|_2^2$.

Assume that $N >> \max(M, P)$, write an Julia function that finds $\widehat{\boldsymbol{W}}$ *efficiently*.

In Julia, your file should be named **rmlls.jl** and should contain the following function:

```
"""
W_hat = rmlls(X, Y, beta)
Returns the solution to the regularized multivariate linear least squares problem.
In:
* X: MxN matrix whose each column is a training feature
* Y: PxN matrix whose each column is a training response
* betea: The regularization weight
Out:
*  W_hat  : matrix of size MxP
For full credit, your solution should be computationally efficient!
"""
function rmlls(X, Y, beta)
```

**Hint 1**: This is similar to the supervised linear least-squares problem HW06 Pr 4, but with the difference that there is an regularization term in the loss and the responses $\boldsymbol{y}_n$ are multidimensional vectors. Can you divide the problem into smaller ones so that each of them resembles the optimization problem in HW06 Pr 4 (with regularization)?

**Hint 2**: You can also solve this problem by matrix differentiation. The matrix derivative of a scalar valued function $f : \mathbb{R}^{M \times P} \to \mathbb{R}$ is given by

$$\nabla_{\boldsymbol{W}} f(\boldsymbol{W}) = \begin{bmatrix} \frac{\partial f}{\partial w_{11}} & \frac{\partial f}{\partial w_{12}} & \cdots & \frac{\partial f}{\partial w_{1P}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial w_{M1}} & \frac{\partial f}{\partial w_{M2}} & \cdots & \frac{\partial f}{\partial w_{MP}} \end{bmatrix}$$

You can find $\widehat{\boldsymbol{W}}$ by solving for $\nabla_{\boldsymbol{W}} L_\beta(\widehat{\boldsymbol{W}}) = 0$.

**Solution 1**

$$\nabla_{\boldsymbol{W}} L_\beta(\boldsymbol{W}) = \boldsymbol{X}\boldsymbol{X}'\boldsymbol{W} - \boldsymbol{X}\boldsymbol{Y}' + 2\beta\boldsymbol{W}$$

Solving for $\nabla_{\boldsymbol{W}} L_\beta(\widehat{\boldsymbol{W}}) = 0$, we get

$$\widehat{\boldsymbol{W}} = (\boldsymbol{X}\boldsymbol{X}' + 2\beta\boldsymbol{I})^{-1}(\boldsymbol{X}\boldsymbol{Y}')$$

**Solution 2** Note the following alternative expression for $L_\beta$ involving block matrices:

$$L_\beta(\boldsymbol{W}) = \left\| \boldsymbol{W}' \left[\boldsymbol{X} \ \ \sqrt{2\beta}\boldsymbol{I}\right] - \left[\boldsymbol{Y} \ \ \boldsymbol{0}_{P\times M}\right] \right\|_F^2$$

Thus, we can use the backslash operator

```
[X sqrt(2*beta)*Matrix(I(M))]' \ [Y zeros(P, M)]'
```

(This is a bit to harder to come up with but we had seen a similar trick in EECS 545.)
The Julia implementations for both are given below:

```
using LinearAlgebra;
"""
W_hat = rmlls(X, Y, beta)
Returns the solution to the regularized multivariate linear least squares problem.
In:
* X: MxN matrix whose each column is a training feature
* Y: PxN matrix whose each column is a training response
* betea: The regularization weight
Out:
*  W_hat  : matrix of size MxP
For full credit, your solution should be computationally efficient!
"""
function rmlls(X, Y, beta)
    ## Note that temp = (X * Y') requires M N P computations
    ## And inv(X*X' + 2*beta*I(M)) * temp requires M M P computations
    ## Totally = M M P + M P N computations
    ## The other way requires M M N + M P N computations which is larger
    M, N = size(X);
    return inv(X*X' + 2*beta*I(M)) * (X * Y')
end

'''
Alternatively you can implement it this way too.
'''
function rmlls(X, Y, beta)
    M, N = size(X);
    P, _ = size(Y);
    return ([X sqrt(2*beta)*Matrix(I(M))])' \ [Y zeros(P, M)]'
end
```