Part1: Constrained Least Squares Problems

1. Linearly constrained least squares

the (linearly) constrained least squares problem (CLS) is

minimize $\|Ax-b\|^2$

subject to $Cx=d$

$\|Ax-b\|^2$ is the objective function and Cx=d are equality constraints, x is an n-vector, A is a m × n matrix, b is an m-vector, C is a p × n matrix, and d is a p-vector.

$\hat{\mathbf{x}}$ is a solution of CLS if $C\hat{x}=d$ and $\|A\hat{x}-b\| \leq \|Ax-b\|^2$ holds for any n-vector x that satisfies $Cx=d$

2.Methods of solving the constrained least squares problem

2.1 KKT equations optimality conditions in matrix-vector form:

$$2\left(A^TA\right)\hat{x}-2A^T\mathbf{b}+C^T\mathbf{z}=0, C\hat{\mathbf{x}}=\mathbf{d}$$

put these together to get KKT conditions $\begin{bmatrix} 2\left(A^TA\right) & C^T \\ C & 0 \end{bmatrix}\begin{bmatrix} \hat{x} \\ \hat{z} \end{bmatrix}=\begin{bmatrix} 2A^Tb \\ d \end{bmatrix}$

then we get $\begin{bmatrix} \hat{\mathrm{x}} \\ \hat{\mathrm{z}} \end{bmatrix}=\begin{bmatrix} 2\left(A^TA\right) & C^T \\ C & 0 \end{bmatrix}^{-1}\begin{bmatrix} 2A^T\mathrm{b} \\ \mathrm{d} \end{bmatrix}$

2.2 QR factorization

3.Least norm problem (an important special case of Least Squares Problems)

minimize $\|x\|^2$

subject to $c_i^Tx=d_i, \mathrm{i}=1,\ldots,\mathrm{p}$

3.1 form Lagrangian function, with Lagrange multipliers
$z_1,\ldots,z_p.\,\mathrm{L(x,z)}=\mathrm{f(x)}+z_1\left(c_1^Tx-d_1\right)+\cdots+z_p\left(c_p^Tx-d_p\right)$

3.2 optimality conditions are
$\frac{\partial \mathrm{L}}{\partial x_i}(\hat{x},\hat{z})=2\sum_{j=1}^{n}\left(A^TA\right)_{ij}\hat{x}_j-2\left(A^Tb\right)_i+\sum_{j=1}^{p}\hat{z}_j\left(c_j\right)_i=0, i=1,\ldots,n, \frac{\partial \mathrm{L}}{\partial z_i}(\hat{x},\hat{z})=c_i^Tx-d_i=0, \mathrm{i}=1,\ldots,\mathrm{p},$

3.3 Lagrange Multipliers = Derivatives of the Cost

Our first example is in two dimensions. The function F is quadratic. The set K is linear.

$$F(x)=x_1^2+x_2^2$$

on the line

$$K: a_1x_1+a_2x_2=b$$

On the line K, we are looking for the point that is nearest to (0,0). The cost F(x) is distance squared. We can discover this from simple calculus, after we bring the constraint equation $a_1x_1+a_2x_2=b$ into the function $F(x)=x_1^2+x_2^2$

Multiply $a_1x_1+a_2x_2-b$ by an unknown multiplier λ and add it to F(x)

Lagrangian $L(x,\lambda) = F(x) + \lambda(a_1 x_1 + a_2 x_2 - b)$

$= x_1^2 + x_2^2 + \lambda(a_1 x_1 + a_2 x_2 - b)$

Set the derivatives $\partial L/\partial x_1$ and $\partial L/\partial x_2$ and $\partial L/\partial \lambda$ to zero. Solve those three equations for x_1, x_2, λ

$$\partial L/\partial x_1 = 2x_1 + \lambda a_1 = 0$$
$$\partial L/\partial x_2 = 2x_2 + \lambda a_2 = 0$$
$$\partial L/\partial \lambda = a_1 x_1 + a_2 x_2 - b$$

We can get

$$\lambda = -2b/(a_1^2 + a_2^2)$$
$$x_1 = -\lambda a_1/2 = a_1 b/(a_1^2 + a_2^2)$$
$$x_2 = -\lambda a_2/2 = a_2 b/(a_1^2 + a_2^2)$$
$$x_1^2 + x_2^2 = b^2/(a_1^2 + a_2^2)$$

The derivative of the minimum cost with respect to the comtrainst level b is minus the Lagrange multiplier:

$$d/db(b^2/(a_1^2 + a_2^2)) = 2b/(a_1^2 + a_2^2) = -\lambda$$

Part2: Solving constrained least problems

1. Constraint least squares via KKT equations

Algorithm

Step 1 : Form Gram matrix. Compute $A^T A$

Step 2 : Solve KKT equations by QR factorization and back substitution.

Time complexity : $mn^2 + 2(n+p)^3$

An example of Constraint least squares via KKT equations in Julia:

Firstly, we randomly generate A, b, C, d

In [1]:

```
m = 10; n = 5; p = 2;
A = randn(m,n); b = randn(m); C = randn(p,n); d = randn(p);
```

Define the function cls_solve_kkt() to implement the algorithm.

In [2]:

```
function cls_solve_KKT(A,b,C,d)
m, n = size(A)
p, n = size(C)
G = A'*A # Gram matrix
KKT = [2*G C'; C zeros(p,p)] # KKT matrix
xzhat = KKT \ [2*A'*b; d]
return xzhat[1:n,:]
end;
```

Test the function cls_solve_kkt().

In [3]:

```
cls_solve_KKT(A,b,C,d)
```

Out[3]:

```
5×1 Array{Float64,2}:
  0.0990152352422455
  0.11870067836607096
 -0.490651246965513
  0.6001981250682451
 -0.22194625122020822
```

2. Constraint least squares via QR factorization

Algorithm

Step 1 : Compute the QR factorizations. $\begin{bmatrix} A \\ C \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \quad Q_2^T = \tilde{Q}\tilde{R}$

Step 2 : Compute $\tilde{R}^{-T} d$ by forward substitution.

Step 3 : Form right-hand side and solve $\tilde{R}w = 2\tilde{Q}^T Q_1^T b - 2\tilde{R}^{-T} d$ via back substitution.

Step 4 : Compute$\hat{x}$ Form right-hand side and solve $R\hat{x} = Q_1^T b - (1/2)Q_2^T w$ by back substitution.

Time complexity : $2(m+p)n^2 + 2np^2$

An example of Constraint least squares via QR factorization in Julia:

Define the function cls_solve_QR() to implement the algorithm.

In [4]:

```
using LinearAlgebra
function cls_solve_QR(A,b,C,d)
m, n = size(A)
p, n = size(C)
Q, R = qr([A; C])
Q = Matrix(Q)
Q1 = Q[1:m,:]
Q2 = Q[m+1:m+p,:]
Qtil, Rtil = LinearAlgebra.qr(Q2')
Qtil = Matrix(Qtil)
w = Rtil \ (2*Qtil'*Q1'*b - 2*(Rtil'\d))
return xhat = R \ (Q1'*b - Q2'*w/2)
end;
```

Test the function cls_solve_QR() and compare the result with cls_solve_KKT().

Must be same!

In [5]:

```
cls_solve_QR(A,b,C,d)
```

Out[5]:

```
5-element Array{Float64,1}:
  0.0990152352422457
  0.11870067836607087
 -0.4906512469655131
  0.6001981250682452
 -0.22194625122020828
```

3. Sparse constrained least squares

To deal with sparse matrices, the simplest way is to replace the QR factorizations with sparse QR factorizations in the previous algorithms. Fortunately, the built-in function qr() can also realize sparse QR factorizations.

An example of solving sparse constraint least squares in Julia:

Notice that unlike cls_solve_KKT(), this function assumes b and d are vectors. The following formulation will generate a sparse set of equations to solve if A and C are sparse.

In [6]:

```
function cls_solve_sparse(A,b,C,d)
m, n = size(A)
p, n = size(C)
bigA = [ zeros(n,n) A' C';
A -I/2 zeros(m,p) ;
C zeros(p,m) zeros(p,p) ]
xyzhat = bigA \ [zeros(n) ; b ; d]
return xhat = xyzhat[1:n]
end;
```

A random formulation of A, b, C, d.

In [7]:

```
m = 100; n = 50; p = 10;
A = randn(m,n); b = randn(m); C = randn(p,n); d = randn(p);
```

Again, compare based on result of the algorithms of computing via KKT equation and QR factorization. We found the two algorithms agree.

In [8]:

```
x1 = cls_solve_KKT(A,b,C,d);
x2 = cls_solve_sparse(A,b,C,d);
norm(x1-x2)
```

Out[8]:

```
8.863201352429109e-15
```

4. Solution of least norm problem

For the least norm problem, the KKT equation is reduced to

$$\begin{bmatrix} 2I & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{z} \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

Step 1 : QR factorization. Compute the QR factorization $C^T = QR$

Step 2 : Compute$\hat{x}$. Solve $R^T y = d$ by forward substitution.

Step 3 : Compute $\hat{x} = Qy$

Time complexity :$2np^2$

Comparison of sloving least norm problem via different methods in Julia:

In julia, the backlash operator can be used to find the solution of equation $Cx = d$. Here is the comparison of the results of solving a least norm problem by different methods.

In [9]:

```
p = 50; n = 500;
C = randn(p,n); d = randn(p);
# Solve using backslash
x1 = C\d;
# Solve using cls_solve, which uses KKT system
x2 = cls_solve_KKT(Matrix{Float64}(I, n, n), zeros(n), C, d);
# Using pseudo-inverse
x3 = pinv(C)*d;
norm(x1-x2)
```

Out[9]:

```
6.832042562912264e-15
```

In [10]:

```
norm(x1-x3)
```

Out[10]:

```
5.904483805221459e-16
```

The result is obvious that the three methods agree when they deal with least norm problem.

Part3: Constrained least squares applications

There are two main applications for constrained least squares, one is Linear quadratic control and another is Linear quadratic state estimation. In the control problem, we can choose the inputs; they are under our control. Once we choose the inputs, we know the state sequence. The inputs are typically actions that we take to affect the state trajectory. In the estimation problem, the inputs (called process noise in the estimation problem) are unknown, and the problem is to guess them.

Application 1: Linear quadratic control

dynamics equations

$$x_{t+1} = A_t x_t + B_t u_t$$
$$y_t = C_t x_t$$

$$J_{\text{output}} = \|y_1\|^2 + \cdots + \|y_T\|^2 = \|C_1 x_1\|^2 + \cdots + \|C_T x_T\|^2$$
$$J_{\text{input}} = \|u_1\|^2 + \cdots + \|u_{T-1}\|^2$$

The linear quadratic control problem (with initial and final state constraints) is

minimize $J_{\text{output}} + \rho J_{\text{input}}$ -> $\|\tilde{A}z - \tilde{b}\|^2$

subject to $x_{t+1} = A_t x_t + B_t u_t$

and $x_1 = x^{\text{init}}, \quad x_T = x^{\text{des}}$

creating vector z which includes all the variable $z = (x_1, \ldots, x_T, u_1, \ldots, u_{T-1})$

$$\tilde{A} = \left[\begin{array}{ccc|ccc} C_1 & & & & & \\ & C_2 & & & & \\ & & \ddots & & & \\ & & & C_T & & \\ \hline & & & \sqrt{\rho}I & & \\ & & & & \ddots & \\ & & & & & \sqrt{\rho}I \end{array}\right]$$

$$\tilde{C} = \left[\begin{array}{ccccc|ccc} A_1 & -I & & & & B_1 & & \\ & A_2 & -I & & & & B_2 & \\ & & \ddots & \ddots & & & & \ddots \\ & & & A_{T-1} & -I & & & B_{T-1} \\ \hline I & & & & & & & \\ & & & & I & & & \end{array}\right], \quad \tilde{d} = \left[\begin{array}{c} 0 \\ 0 \\ \vdots \\ 0 \\ \hline x^{\text{init}} \\ x^{\text{des}} \end{array}\right].$$

Example

In [11]:

```
H = randn(2,2); # creating 2*2 matrix
```

In [12]:

```
using LinearAlgebra
identity_matrix=Matrix{Float64}(I, 3, 3);# creating 3*3 identity matrix
```

In [13]:

```
kron(identity_matrix,H) # using kronecker function to block diagonal matrix
```

Out[13]:

```
6×6 Array{Float64,2}:
  1.04291   0.0107771   0.0       0.0         0.0       0.0
 -0.534264  0.578265   -0.0       0.0        -0.0       0.0
  0.0       0.0         1.04291   0.0107771   0.0       0.0
 -0.0       0.0        -0.534264  0.578265   -0.0       0.0
  0.0       0.0         0.0       0.0         1.04291   0.0107771
 -0.0       0.0        -0.0       0.0        -0.534264  0.578265
```

In [14]:

```
function cls_solve(A,b,C,d)
m, n = size(A)
p, n = size(C)
Q, R = qr([A; C])
Q = Matrix(Q)
Q1 = Q[1:m,:]
Q2 = Q[m+1:m+p,:]
Qtil, Rtil = qr(Q2')
Qtil = Matrix(Qtil)
w = Rtil \ (2*Qtil'*Q1'*b - 2*(Rtil'\d))
return xhat = R \ (Q1'*b - Q2'*w/2)
end;
```

In [15]:

```
function eye(k)
    matrix=I+zeros(k,k)
return matrix
end;
```

In [16]:

```
function lqr(A,B,C,x_init,x_des,T,rho)
n = size(A,1)
m = size(B,2)
p = size(C,1)
q = size(x_init,2)
Atil = [ kron(eye(T), C) zeros(p*T,m*(T-1)) ;zeros(m*(T-1), n*T) sqrt(rho)*eye(m*(T-1)) ]
btil = zeros(p*T + m*(T-1), q)
# We'll construct Ctilde bit by bit
Ctil11 = [ kron(eye(T-1), A) zeros(n*(T-1),n) ] -[ zeros(n*(T-1), n) eye(n*(T-1)) ]
Ctil12 = kron(eye(T-1), B)
Ctil21 = [eye(n) zeros(n,n*(T-1)); zeros(n,n*(T-1)) eye(n)]
Ctil22 = zeros(2*n,m*(T-1))
Ctil = [Ctil11 Ctil12; Ctil21 Ctil22]
dtil = [zeros(n*(T-1), q); x_init; x_des]
z = cls_solve(Atil,btil,Ctil,dtil)
x = [z[(i-1)*n+1:i*n,:] for i=1:T]
u = [z[n*T+(i-1)*m+1 : n*T+i*m, :] for i=1:T-1]
y = [C*xt for xt in x]
return x, u, y
end;
```

$$x = [x[1], x[2], x[3], x[4], x[5], \ldots\ldots, x[T]]$$

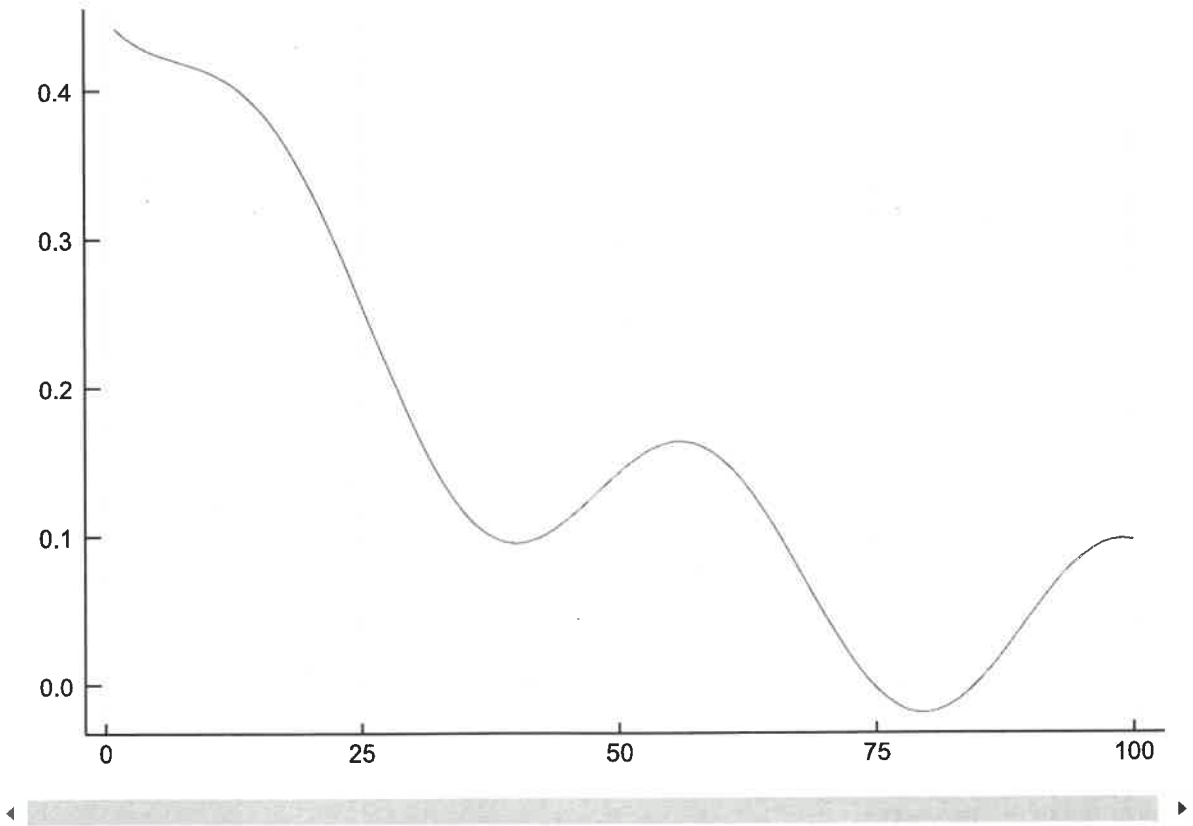
$$u = [u[1], u[2], u[3], u[4], u[5], \ldots\ldots, u[T]]$$

$$y = [y[1], y[2], y[3], y[4], y[5], \ldots\ldots, y[T]]$$

In [17]:

```
using LinearAlgebra
A = [ 0.855 1.161 0.667;
0.015 1.073 0.053;
-0.084 0.059 1.022 ];
B = [-0.076; -0.139; 0.342 ];
C = [ 0.218 -3.597 -1.683 ];
n = 3; p = 1; m = 1;
x_init = [0.496; -0.745; 1.394];
x_des = zeros(n,1);
T = 100;
yol = zeros(T,1);
Xol = [ x_init zeros(n, T-1) ];
for k=1:T-1
Xol[:,k+1] = A*Xol[:,k];
end;
yol = C*Xol;
using Plots
plot(1:T, yol', legend = false)
```

Out[17]:



$$x = [x[1], x[2], x[3], x[4], x[5], \ldots\ldots, x[T]]$$

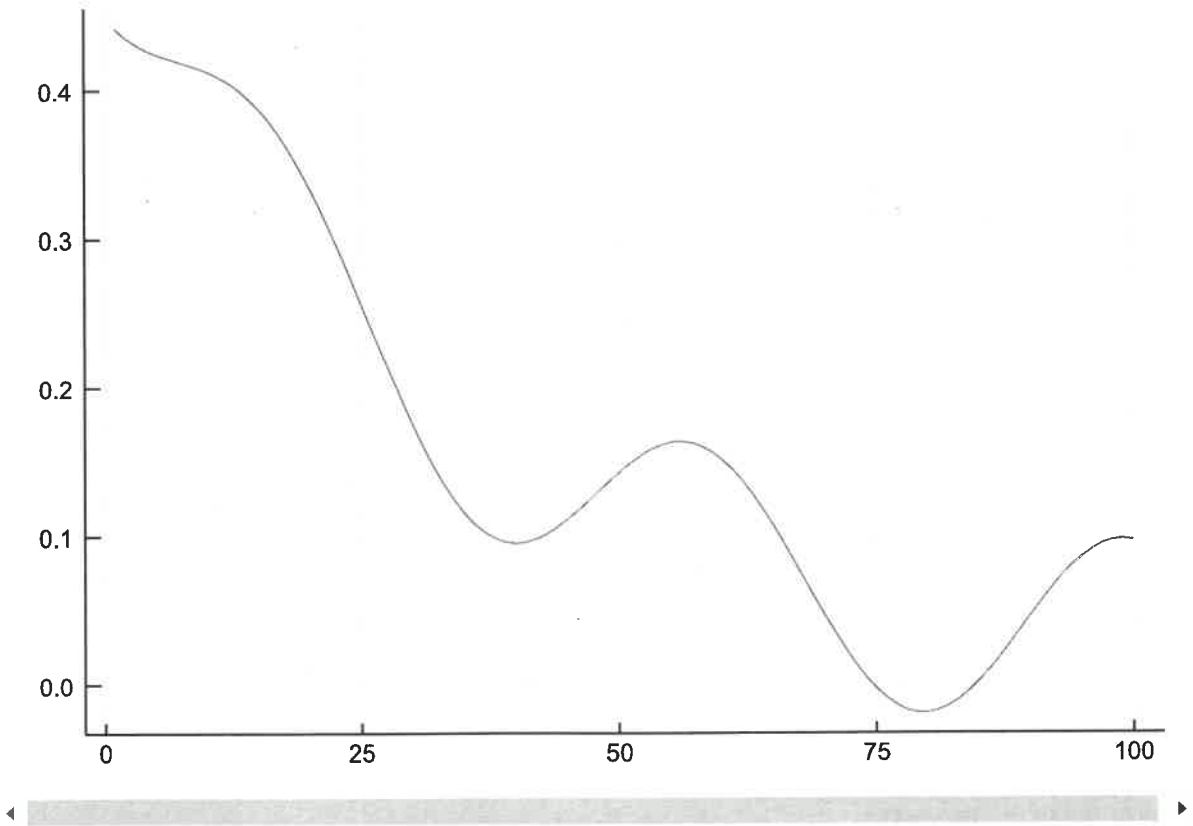
$$u = [u[1], u[2], u[3], u[4], u[5], \ldots\ldots, u[T]]$$

$$y = [y[1], y[2], y[3], y[4], y[5], \ldots\ldots, y[T]]$$

In [17]:

```
using LinearAlgebra
A = [ 0.855 1.161 0.667;
0.015 1.073 0.053;
-0.084 0.059 1.022 ];
B = [-0.076; -0.139; 0.342 ];
C = [ 0.218 -3.597 -1.683 ];
n = 3; p = 1; m = 1;
x_init = [0.496; -0.745; 1.394];
x_des = zeros(n,1);
T = 100;
yol = zeros(T,1);
Xol = [ x_init zeros(n, T-1) ];
for k=1:T-1
Xol[:,k+1] = A*Xol[:,k];
end;
yol = C*Xol;
using Plots
plot(1:T, yol', legend = false)
```

Out[17]:



In [18]:

```
rho = 0.2;
T = 100;
x, u, y = lqr(A,B,C,x_init,x_des,T,rho)
J_input = norm(u)^2
```

Out[18]:

```
0.7738942551160125
```

In [19]:

```
J_output = norm(y)^2
```

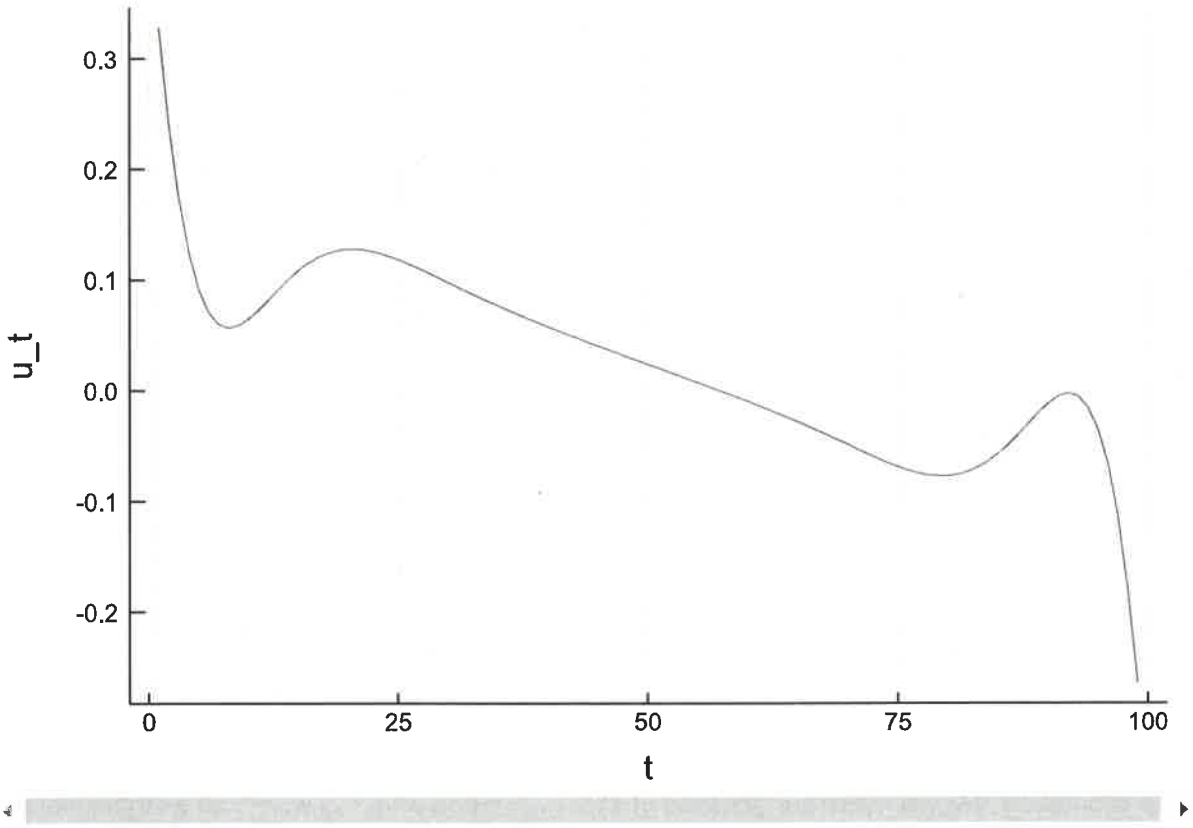
Out[19]:

```
3.7829986463324596
```

In [20]:

```
plot(1:T-1, vcat(u...), legend = false, xlabel="t",ylabel= "u_t")
```

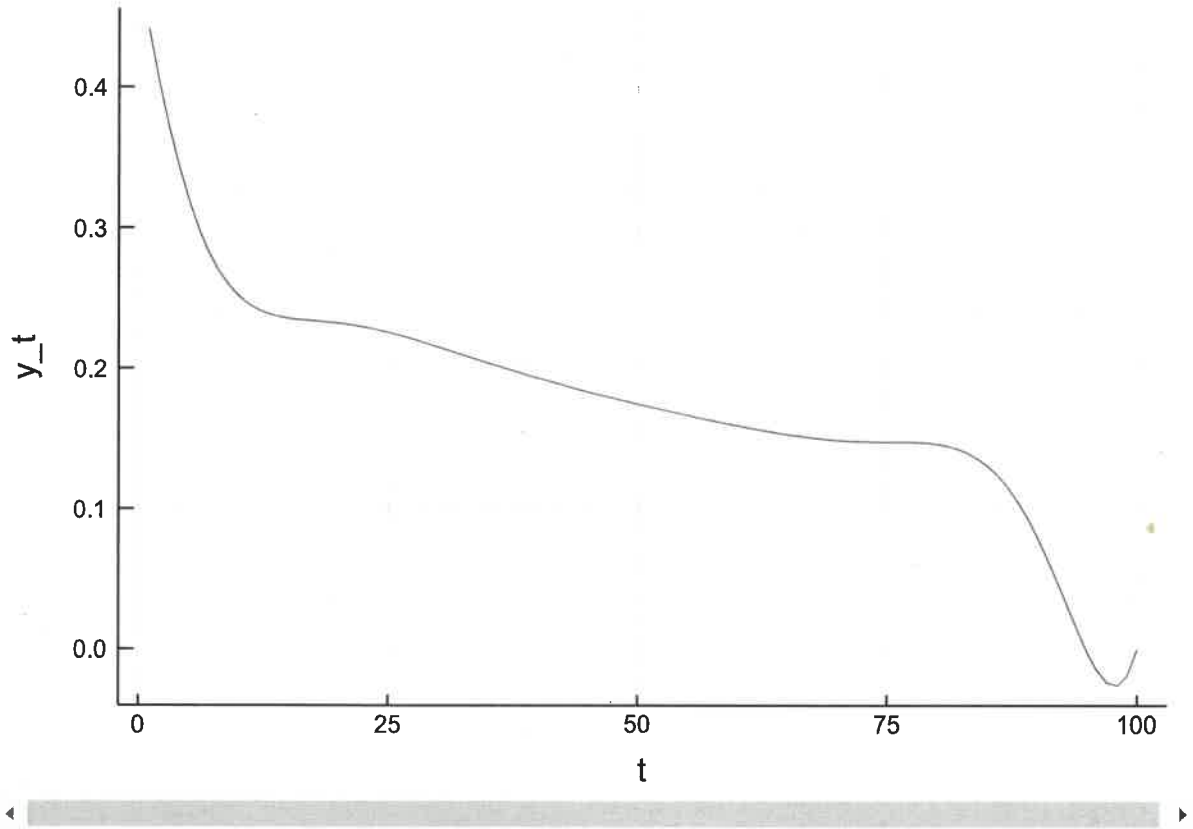
Out[20]:



In [21]:

```
plot(1:T, vcat(y...), legend=false, xlabel = "t",ylabel = "y_t")
```

Out[21]:



Application2: Linear quadratic state estimation

Linear dynamical system equations

$x_{t+1} = A_t x_t + B_t w_t$

$y_t = C_t x_t + v_t$

$J_{\text{meas}} = \|v_1\|^2 + \cdots + \|v_T\|^2 = \|C_1 x_1 - y_1\|^2 + \cdots + \|C_T x_T - y_T\|^2$

$J_{\text{proc}} = \|w_1\|^2 + \cdots + \|w_{T-1}\|^2$

Least squares state estimation. We will make our guesses of x1, . . . , xT and w1, . . . , wT −1 so as to minimize a weighted sum of our objectives, subject to the dynamics constraints:

minimize $J_{\text{meas}} + \lambda J_{\text{proc}}$

subject to $x_{t+1} = A_t x_t + B_t w_t$

Type *Markdown* and LaTeX: α^2