# 广义预测控制实验报告


> 姓名：张千一
> 学号：1120210190
> 日期：2021.12.10
> 邮箱：1120210190@mail.nankai.edu.cn


## 广义预测控制简介

广义预测控制(Generalized Predictive Control, GPC)是模型预测控制的经典方法，可在历史信息的输入下，对较准确**未来预测**，并进行**模型的自校正**。它是Clarke等在1987年提出的基于参数模型的控制算法，它是以受控自回归积分滑动平均(CARIMA)模型为基础，并结合了辨识和自校正机制，具有良好的鲁棒性。对于无约束或仅含软约束的广义预测控制中，可以对输入输出进行偏差处理，进而得到闭式解，使得运算较快。实时性、鲁棒性和自校正能力使得GPC在控制、机器人等领域具有广泛的应用前景。

其算法分为四个主要模块：

- 预测模型：利用准确模型/拟合模型进行前向推导，获得理想情况（无噪声）下的预测值。
- 跟踪轨迹：考虑目标值与当前值间的偏差，将目标值在时间维度上离散柔滑为目标轨迹。
- 滚动优化：确定优化的目标，由“轨迹跟踪误差”与“控制量的变化”两部分组成
- 反馈校正：对于有准确模型的系统，不需要使用反馈校正；对于无准确模型的系统，需要利用历史数据和最小二乘法，拟合模型参数。

无约束的GPC利用了丢番图和二次型优化的特点，具有如下优点：

- 采用CARIMA模型作为预测模型。该模型比较接近实际对象特性，而且具有积分作用，能有效清楚静态误差，参数数目少，适合在线实现，对模型阶次不甚敏感。
- 引入丢番图方程，并采用地推算法求解，节省了计算时间
- 采用有限时域的长时段多步预测，使GPC适用带负载扰动、未知或变滞后的被控对象。
- 采用对输出误差和控制增强加权的二次型性能指标，有利于提高闭环系统的稳定性。引入控制时域的概念，减小了计算量。

## GPC的算法流程与数学推导

为了逻辑清晰，用“模型预测”、“跟踪轨迹”、“模型转换”、“滚动优化”、“反馈校正”五个步骤来进行说明。

### 预测模型

整个系统基于CARIMA模型，公式如图所示：

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + \xi/\Delta \tag{1}$$

> 其中，$z^{-1}$是后移时间算子，即$z$变化的逆变换(拉普拉斯变换)。$y(k)$是当前时刻的系统状态(输出)，可直接观测得到。$u(k-1)$是上一时刻的输入。$\xi$是噪声干扰。$\Delta = 1 - z^{-1}$是差分算子，本质上是当前时刻与上一时刻的对应量的差值。

$A(z^{-1})$和$B(z^{-1})$是系统的在上一时刻的模型，对与模型已知的GPC来说，这两项是常数；对于模型未知的GPC来说，他们是上一时刻拟合出来的模型参数。

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + \ldots + a_{na} z^{-na} \\ B(z^{-1}) &= b_0 + b_1 z^{-1} + \ldots + b_{nb} z^{-nb} \end{aligned} \tag{2}$$

> 其中，$\delta A(z^{-1}) = na$是$A$的可变参数个数，$\delta B(z^{-1}) = nb$，$nb+1$是$B$的可变参数个数。注意，$a_0$=1是固定值。

### 跟踪轨迹

在当前时刻k，系统具有自己所处的状态y(k)。此时，若外界给定一个新目标值$y_r(k)$（数值），该系统需要快速且稳定的到达该目标值。如果目标值与当前值相差较大，系统所需的控制量极大，损害电机。于是，我们需要对目标值进行柔化，产生子目标序列或叫做跟踪轨迹$y_d(k)$（长度为N），帮助系统稳定的到达目标点。

$$\begin{gathered} y_d(k) = y(k) \\ y_d(k+j) = \alpha y_d(k+j-1) + (1-\alpha) y_r(k) \quad j \in [1, N] \end{gathered} \tag{3}$$

> 其中，\alpha是柔化因子，$N$是预测长度。柔化是个自举的过程。

### 模型转换

在前两节中，介绍了系统模型和需要跟踪的轨迹。接下来，立足于当前时刻k和需要目标状态$y_r(k)$，系统需要 **计算未来N个时刻应该执行的控制量**$u_j$，进而尽可能让系统在$k+j$时刻达到$y_d(k+j)$。这是一个软约束下的线性优化问题，可以利用求解器求解，但为了加快速度，我们对模型进行了转换，将输入与输出的关系式转换为 **输入变化与输出** 的关系式，进而利用闭式解（矩阵运算）直接求得 **最优控制量**。

回顾公式(1)，模型参数$A$和$B$已知时，给定输入，即可得到下一时刻的输出。于是，我们可以利用自举，假设$k+j$时刻的输入为$u_{k+j}$，即可得到$k+j+1$时刻的输出$y(k+j+1)$。对于未来$N$步，每一步都可利用(1)得到一个等式。对于这$N$个等式，我们可以通过一定的转化，使其变成如下形式：

$$1 = E_j(z^{-1}) A(z^{-1}) \Delta + z^{-j} F_j(z^{-1}), \quad j \in [1, N] \tag{4}$$

其中$E_j$和$F_j$的格式如下：

$$\begin{gathered} E_j = e_0 + e_1 z^{-1} + \ldots + e_{j-1} z^{-(j-1)} \\ F_j = f_{j,0} + f_{j,1} z^{-1} + \ldots + f_{j,na} z^{(-na)} \end{gathered} \tag{5}$$

> 注意，E的后项充分利用了前向，$E_{j+m}$=$E_{j+m+n}$的前$m$项。而F不具备在时间上的连续利用特性。

公式(4)实际上是一个丢番图方程，$E_j$和$F_j$可由如下迭代运算得到：

$$\begin{gathered} E_1 = 1,\ F_1 = z[1 - A\Delta] \\ e_j = f_{j,0} \\ E_{j+1} = E_j + e_j z^{-j},\ F_{j+1} = z[F_j - e_j A \Delta], \quad j = [1, N] \end{gathered} \tag{6}$$

> 其中，第一行进行了初始化，第二三行迭代执行求解下一时刻的值

从另一个角度处理公式(1)，为了得到矩阵形式的前向预测值，我们在(1)两边同时乘以$E_j \Delta z^j$，可得

$$E_j A(z^{-1}) \Delta y(k+j) = E_j B(z^{-1}) \Delta u(k+j-1) + E_j \xi(k+j) \tag{7}$$

将公式(4)带入公式(7)可得：

$$y(k+j) = E_j B(z^{-1}) \Delta u(k+j-1) + F_j y(k) + E_j \xi(k+j) \tag{8}$$

公式(8)是精确模型，但由于噪声$\xi$微小、不可观测且无法建模，所以这里假设噪声不存在，得到预测值为：

$$\hat{y}(k+j|k) = E_j B(z^{-1}) \Delta u(k+j-1) + F_j y(k) \tag{9}$$

> 其中，等号右边除待求量\Delta u(k+j-1)外，其他均已知。

为了求解方便，我们再次进行转换，考虑另一组丢番图：

$$E_j B(z^{-1}) = G_j + z^{-j} H_j \quad j \in [1, N] \tag{10}$$

其中，$G_j$和$H_j$的格式如下：

$$\begin{gathered} G_j = g_0 + g_1 z^{-1} + \ldots + g_{j-1} z^{-(j-1)} \\ H_j = h_{j,0} + h_{j,1} z^{-1} + \ldots + h_{j,nb-1} z^{-(nb-1)} \end{gathered} \tag{11}$$

> 其中，$\delta E_jB(Z^{-1})=j$，$\delta G_j=j-1$，$\delta H_j=nb-1$。实际上，$G_j$和$H_j$是在不同维度(变换算子)下对$E_jB(Z^{-1})$进行简单的拆分。另外，正如在(5)中所说，E_j利用了前向信息，所以$G_j$也具有时间上的连续特性。

将(10)带入(9)有：

$$\hat{y}(k+j|k)=G_j\Delta u(k+j-1)+H_j\Delta u(k-1)+F_jy(k) \tag{12}$$

对(10)进行简化，令$y_0(k+j)=H_j\Delta u(k-1)+F_jy(k)$，有：

$$\hat{y}(k+j|k)=G_j\Delta u(k+j-1)+y_0(k+j) \tag{13}$$

公式(13)对于每个时刻$k+j$均成立，对与$j\in[1,N]$，可将其写成矩阵形式：

$$\hat{Y}(k)=Y_0(k)+G^*\Delta U^*(k) \tag{14}$$

其中：

$$G^*=\begin{bmatrix} g_0 & & & \\ g_1 & g_0 & & \\ \dots & \dots & \dots & \\ g_{N-1} & g_{N-2} & g_{N-3} & \dots g_0 \end{bmatrix} \tag{15}$$

引入控制步长$N_u\le N$，假设$j>N_u$时，控制量变化$\Delta u(k+j-1)=0$，(15)可改写成：

$$\hat{Y}(k)=Y_0(k)+G\Delta U(k) \tag{16}$$

其中：

$$G=\begin{bmatrix} g_0 & & \\ g_1 & g_0 & \\ \dots & \dots & \dots \\ g_{N_u-1} & \dots & g_0 \\ g_{N_u} & \dots & g_1 \\ \dots & \dots & \dots \\ g_{N-1} & g_{N-2} & \dots g_{N-N_u} \end{bmatrix} \tag{17}$$

$$U=\begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \dots \\ \Delta u(k+N_u+1) \end{bmatrix} \tag{18}$$

## 滚动优化

定义目标函数为：

$$J=\sum_{j=1}^{N}[\hat{y}(k+j|k)-y_d(k+j)]^2+\lambda\sum_{j=1}^{N_u}[\Delta u(k+j-1)]^2 \tag{19}$$

> 其中，第一项考虑了按照预估的控制量，未来时刻到达的状态与柔化状态的差值；第二项考虑了控制量的变化程度。λ是控制加权因子

将(18)写成向量形式有：

$$J=[\hat{Y}(k)-Y_d(k)]^T[\hat{Y}(k)-Y_d(k)]+\lambda[\Delta U(k)]^T[\Delta U(k)] \tag{20}$$

对(19)求极值$\frac{\delta J}{\delta\Delta U(k)}=0$，得到控制律：

$$\Delta U(k)=(G^TG+\lambda I)^{-1}G^T[Y_d(k)-Y_0(k)] \tag{21}$$

取第一个分量作为当前控制量的改变量，得到控制输入应为：

$$u(k)=u(k-1)+\Delta u(k) \tag{22}$$

### 反馈校正

若模型已知，则不需要利用反馈校正。若系统模型参数$A$和$B$未知，则需要利用最小二乘法进行在线识别。设系统的模型为：

$$\begin{gathered}\hat{A}(z^{-1})y(k) = \hat{B}(z^{-1})u(k-1) + \xi/\Delta \\ \Delta y(k) = [1 - \hat{A}(z^{-1})]\Delta y(k) + \hat{B}(z^{-1})\Delta u(k-1) = \hat{\theta}^T(k)\phi(k)\end{gathered} \tag{23}$$

> 其中，$\hat{\theta}(k)$是在整合未知的模型参数；$\phi(k)$是在整合历史数据

$$\begin{gathered}\hat{\theta}(k) = [-\hat{a_1}, \ldots, -\hat{a_{na}}, \hat{b_0}, \ldots, \hat{b_{nb}}] \\ \phi(k) = [\Delta y(k-1), \ldots, \Delta y(k-na), \Delta u(k-1), \ldots, \Delta u(k-nb)]^T\end{gathered} \tag{24}$$

用户给定初值$\hat{\theta}(0) = \theta_0$和$P(0) = \beta^2 I$后，用最小二乘法进行在线辨识：

$$\begin{gathered}\hat{\theta}(k) = \hat{\theta}(k-1) + \frac{P(k-1)\phi(k)}{1 + \phi^T(k)P(k-1)\phi(k)}[\Delta y(k) - \hat{\theta}^T(k-1)\phi(k)] \\ P(k) = P(k-1) - \frac{P(k-1)\phi(k)\phi^T(k)P(k-1)}{1 + \phi^T(k)P(k-1)\phi(k)}\end{gathered} \tag{25}$$

GPC的流程图如下所示：

给定一些初值：$\theta_0, \beta, N, N_u, \lambda, \alpha$

1） $k$ 时刻到来，采样获取 $y(k)$；

2） 用公式(2. 3. 22)、(2. 3. 23)在线辨识 $\hat{A}(z^{-1}), \hat{B}(z^{-1})$ 一次；

3） 用 $\hat{A}(z^{-1}), \hat{B}(z^{-1})$ 代替 $A(z^{-1}), B(z^{-1})$，计算 $E_j, F_j, G_j, H_j$，
进而计算 $Y_0(k), G, (G^T G + \lambda I)^{-1}$；

4） 用(2. 3. 15)计算 $Y_d(k)$；

5） 用(2. 3. 18)、(2. 3. 19)求解控制量 $u(k)$；

6） 数据回归，返回 1）。

如果系统模型 $A, B$ 已知，则 2)省略，3)只需计算一次。

## 代码架构

构建了env这个class，用于模拟真实环境，其中关键函数如下：

```
class env:
    def drawWindow(self)                        # 初始化并打开GUI界面
    def generateTatget(self)                    # 根据GUI选择的信号类型，生成模拟信号
    def hit(self)                               # 读取GUI数据，模拟时间流逝，初始化GPC模型
    def calOutput(self, Yk, has_model=True)     # 调用GPC模型进行求解
    def controlWithU(self, u=None)              # 观测当前时刻的系统状态
```

其中，drawWinwod()的关键代码如下：

```
# 用tkinter和canvas创建GUI界面
window=self.window
self.f = Figure(figsize=(8,3))
self.canvas = FigureCanvasTkAgg(self.f, master=self.window)
# 创建target模块
target_frame = tk.Canvas(window,height=200,width=180)
# 创建model模块
model_frame = tk.Canvas(window,height=200,width=180)
# 创建param模块
param_frame = tk.Canvas(window,height=200,width=180)
# 创建algo模块
algo_frame = tk.Canvas(window,height=140,width=180)
# 创建run按键
but=tk.Button(window,text='run',width=6,height=2,bg='white', font=('Arial', 16, 'bold'),
command=self.hit)
but.place(x=845,y=560,anchor='center')
```

hit()在获取GUI数据后，会以如下逻辑调用另一个class GPC：

```
self.generateTatget()   # 根据信号的type，产生模拟信号
self.createGPC()        # 创建GPC模型
for value in self.targetSignal:
    # 模拟时间流逝，迭代产生控制量
    self.setTarget(value)
    Yk = self.getCurrentY()
    if self.algo_type == "constant":
        u = self.calOutput(Yk, has_model=True)  # 有模型控制
    if self.algo_type == "self-correct":
        u = self.calOutput(Yk, has_model=False) # 无模型控制
    self.controlWithU(u)
self.show()
```

构建了GPC这个class，来在时刻k进行广义预测控制，关键函数如下：

```
def setTargetTraj(self, target_state=10)    # 设定目标信号
def setModel(self, A=[1,-1.5,0.7], B=[1,0.5])   # 有模型控制时被调用，设置模型
def calModel(self, his_y, his_u)    # 无模型控制时被调用，读取历史状态
def calU(self)  # 计算E,F,G,H，进而计算控制量u
```

其中，calModel()进行反馈校正，关键代码如下：

```
# 更新A和B
phy = []
for i in range(self.na):
    phy.append(his_y[-i-1]-his_y[-i-2])
    for i in range(self.nb+1):
        phy.append(his_u[-i-1]-his_u[-i-2])
    phy = np.array(phy).T
    self.theta_k += (self.Yk-his_y[-1] - np.dot(self.theta_k.T,phy)) / (1+np.dot(np.dot(phy.T,self.p_k),phy)) * np.dot(self.p_k,phy)    # 注意这里用的Yk是真实值（可观测）
    self.p_k -= np.dot(np.dot(np.dot(self.p_k,phy),phy.T),self.p_k) /(1+np.dot(np.dot(phy.T,self.p_k),phy))
    self.A = [1.0]
    self.B = []
    for i,value in enumerate(self.theta_k):
        if i<self.na:
            self.A.append(-value)
        else:
            self.B.append(value)
```

calU()通过计算E,F,G,H，进而计算控制量u

```
    def calU(self):
        """
        计算G和H，进而计算输出控制量
        """
        # A_minus1: e^{-1}*A
        self.A_minus1 = [0]
        for value in self.A:
            self.A_minus1.append(value)
        # A_overline = A*DELTA = A*(1-e^{-1})=A-A_minus
        self.A_overline = [-value for value in self.A_minus1]
        for i in range(len(self.A)):
            self.A_overline[i] += self.A[i]
        # E1 and F1
        E = [[1]]
        temp_F = []
        for i,value in enumerate(self.A_overline):
            if i!=0:
```

```
                temp_F.append(-value)
        F = [temp_F]
        # E_{j+1} and F_{j+1}
        for j in range(0,self.N):
            # temp_E
            e_j=F[j][0]
            temp_E = E[j].copy()
            temp_E.append(e_j)
            E.append(temp_E)
            # temp_F
            temp_F = [0 for i in range(max(len(F[j]),len(self.A_overline)))]
            for i,value in enumerate(F[j]):
                temp_F[i]+= value
            for i,value in enumerate(self.A_overline):
                temp_F[i] += -value*e_j
            temp_F_shift = []
            for i,value in enumerate(temp_F):
                if i!=0:
                    temp_F_shift.append(value)
            F.append(temp_F_shift)
        # Ej_B
        Ej_B=[]
        for j in range(0,self.N):
            temp_Ej_B = [0 for i in range(max(len(E[j])+1,len(self.B)))]
            for i,value in enumerate(E[j]):
                temp_Ej_B[i] += value*self.B[0]
                temp_Ej_B[i+1] += value*self.B[1]
            Ej_B.append(temp_Ej_B)
        # G_j & H_j
        G_j = []
        H_j = []
        for j in range(0,self.N):
            G_j=Ej_B[j][:j+1]
            H_j.append(Ej_B[j][j+1])
        # y_0
        Y_k_plus_j = []
        for j in range(0,self.N):
            temp_Fj_y = 0
            for i,value in enumerate(F[j]):
                if i==0:
                    temp_Fj_y += value*self.Yk
                else:
                    temp_Fj_y += value*self.init_y[i-1]
            temp_Y_k_plus_j = H_j[j]*(self.init_u[0]-self.init_u[1]) + temp_Fj_y
            Y_k_plus_j.append(temp_Y_k_plus_j)
        # G
        G = np.zeros([self.N,self.Nu])
        for j in range(self.N):
            for i in range(self.Nu):
                if i+j>=self.N:
                    break
                G[j+i][i]=G_j[j]
        G=np.array(G)
        # combineation
        res = np.linalg.inv(np.dot(G.T,G)+self.lamda*np.array(np.eye(len(G.T))))
        # delta_y
        delta_y = []
        for i in range(len(self.soft_traj)):
            delta_y.append(self.soft_traj[i]-Y_k_plus_j[i])
        delta_Uk = np.dot(np.dot(res,G.T),np.array(delta_y))
```

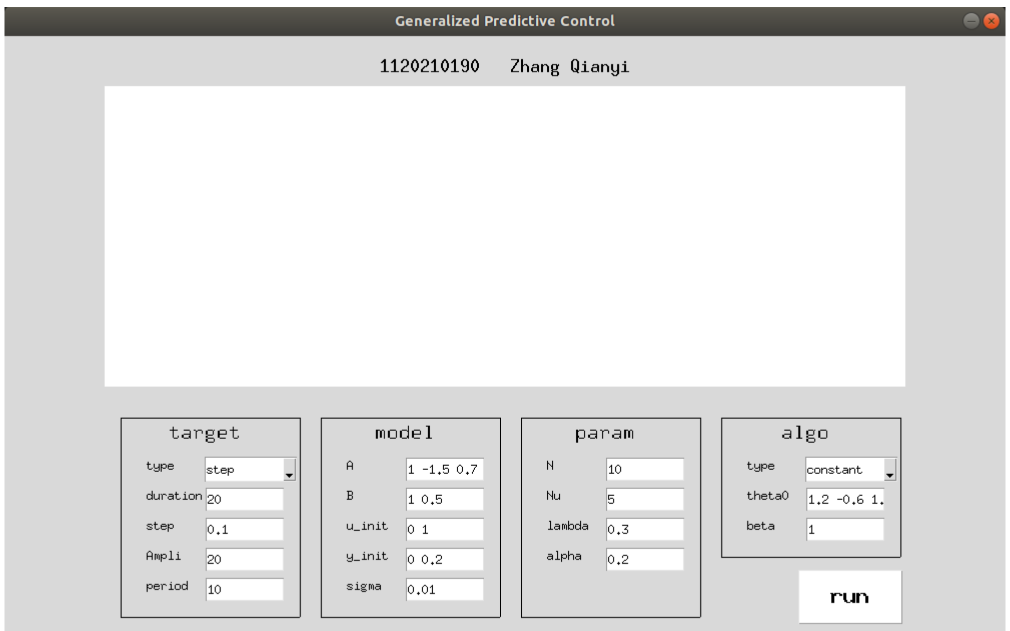
```python
        Y_prdict = np.array(Y_k_plus_j) + np.dot(G,delta_Uk)
        return self.init_u[0] + delta_Uk[0]
```

# 仿真结果

## 仿真界面

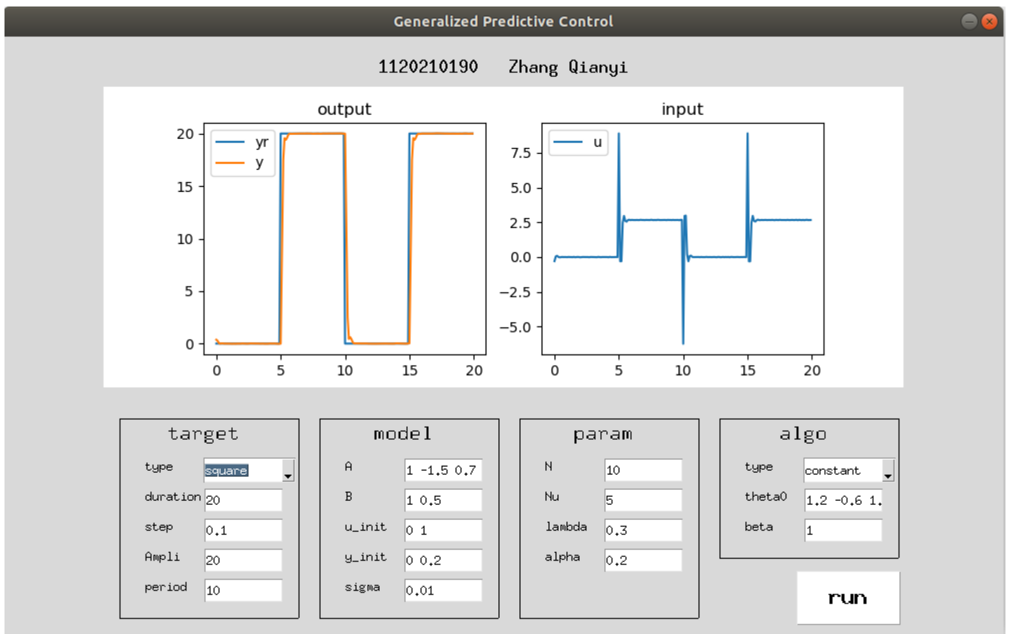
仿真界面如下图所示，其中英文释义如下：

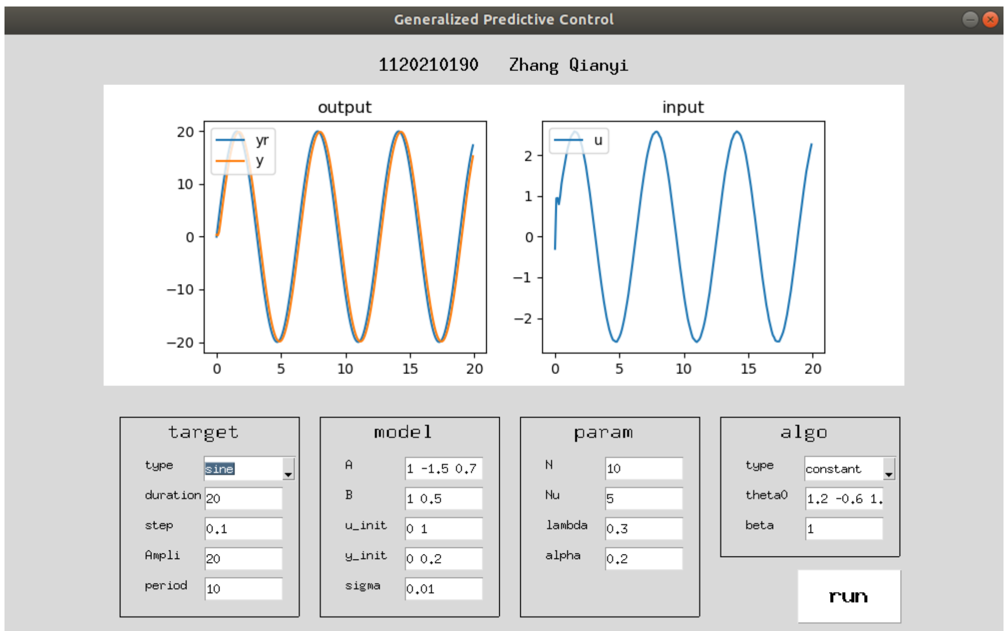
- target模块：
  - type：仿真信号类型
  - duration：仿真时间
  - step：仿真步长
  - Ampli：幅值
  - period：周期
- model模块：
  - A：模型参数A
  - B：模型参数B
  - u_init：历史时刻的控制量
  - y_unit：历史时刻的输出
  - sigma：噪声干扰信号的最大值
- param模块：
  - N：预测步长
  - N_u：控制步长
  - lambda：控制权重因子
  - alpha：柔化因子
- algo模块：
  - type：模型参数是否已知
  - theta0：参数初值
  - beta：参数初值




## 模型已知的仿真结果

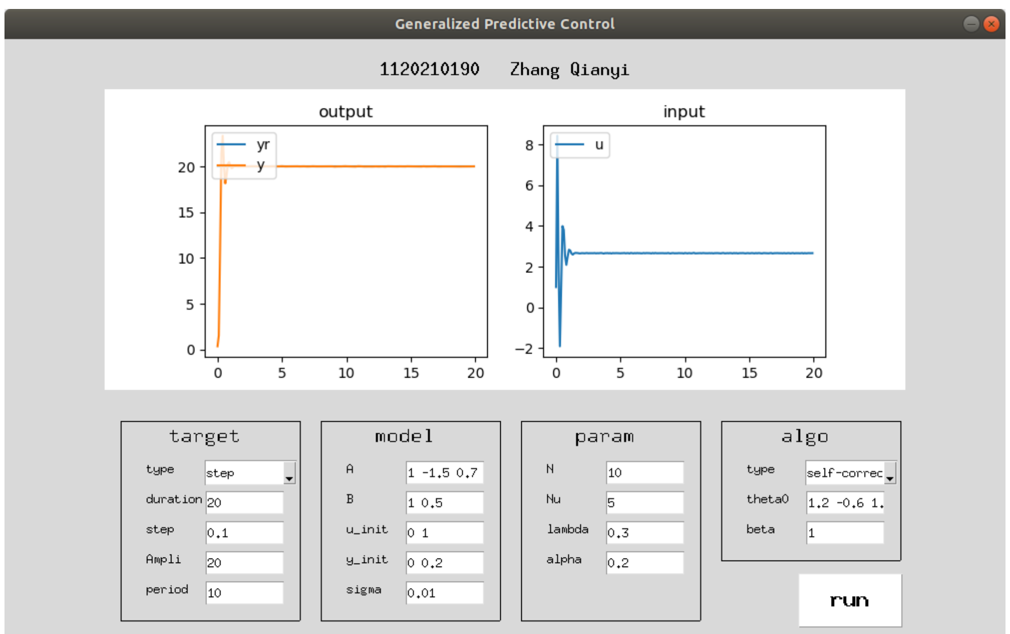
输入分别为阶跃信号，方波信号，正弦信号

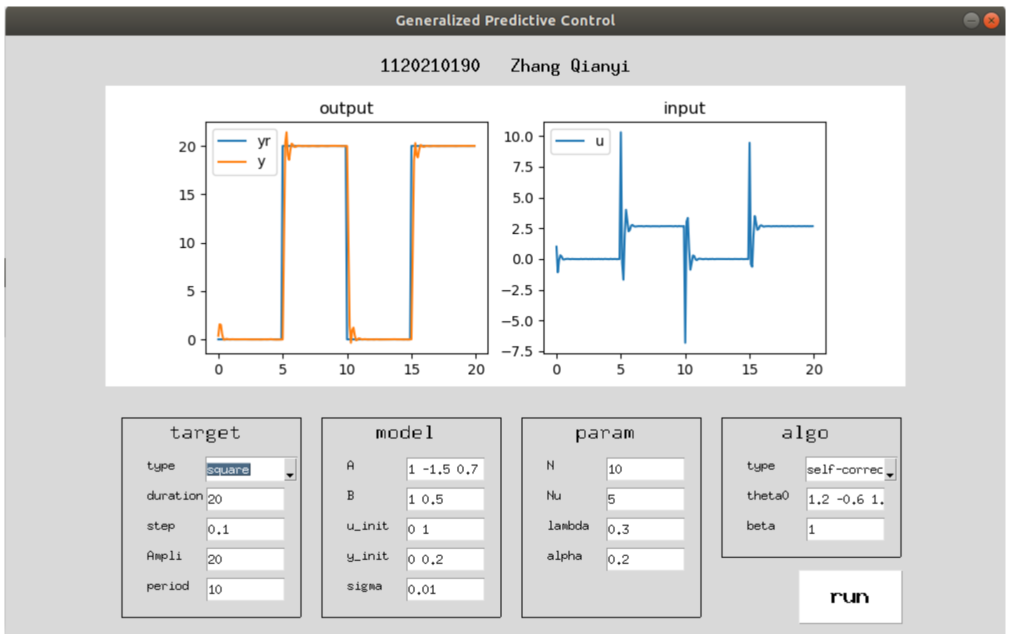



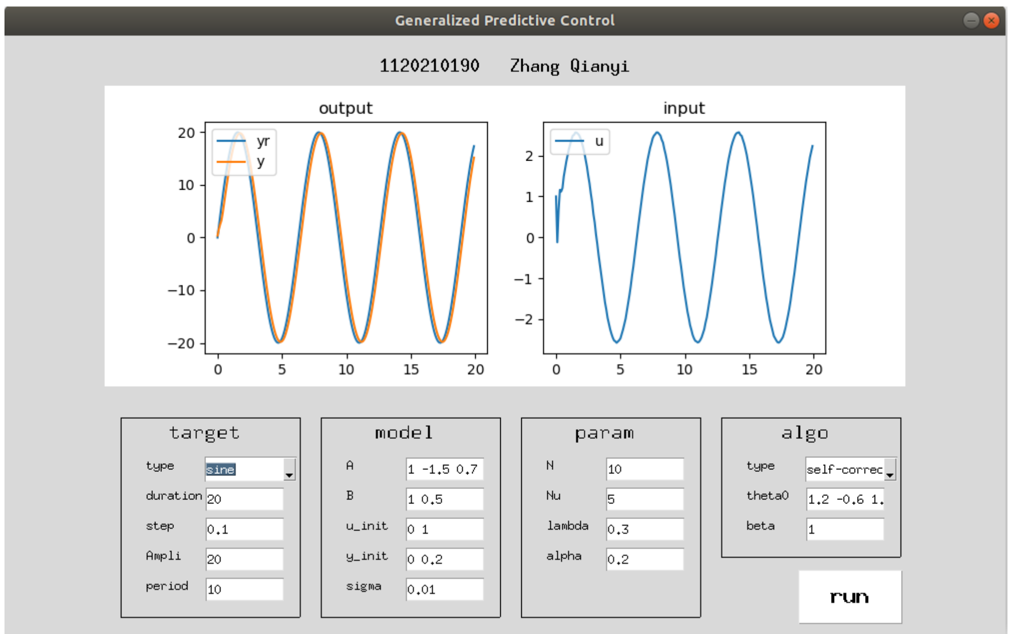




## 模型未知的仿真结果

输入分别为阶跃信号，方波信号，正弦信号






## 参考文献

[1] 陈增强.智能预测控制讲义[M].天津:南开大学.