# Идентификация объекта управления методом наименьших квадратов.

По результатам экспериментальных данных коэффициенты полиномов знаменателя A и числителя B разностной передаточной функции объекта управления можно получить следующим образом:

$$\begin{vmatrix} a_1 \\ a_2 \\ .. \\ b_1 \\ b_2 \\ \\ b_m \end{vmatrix} = |X^T X|^{-1} X^T Y$$

Где X — матрица размером (2*M)xN, полученная из матрицы наблюдения за объектом размером 2x(M+N), состоящей из двух столбцов:

V — входные значения;

Z — выходные значения, полученные через равные интервалы времени.

Y — столбец размером N, являющийся подмножеством столбца Z, в котором пропущены первые M элементов.

M — это порядок модели объекта.

A. Реализация кода и оценка эффективности полученного решения.

1. Написать программу, реализующую представленную методику для чисел с плавающей точкой двойной точности. Функции умножения матриц взять из лабораторного практикума. Функцию умножения матрицы на вектор-столбец реализовать самостоятельно. Код для получения инверсной матрицы методом Гаусса-Жордана можно найти в Интернете:

http://www.programming-techniques.com/2011/09/numerical-methods-inverse-of-nxn-matrix.html

В результате получится следующий код:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <vector>
#include <malloc.h>
```

```
void mat_mul_vec(double **a, double *b, double *c, int N, int M){
   for(int n=0;n<N;n++){
      c[n]=0;
      for(int m=0;m<M;m++){
         c[n]+=a[n][m]*b[m];
      }
   }
}

void mat_mul(double **a, double **b, double **c, int N, int K, int M){
   for(int n=0;n<N;n++){
      for(int m=0;m<M;m++){
         c[n][m]=0.0;
         for(int  k=0;k<K;k++){
            c[n][m] += a[n][k]*b[k][m];
         }
      }
   }
}

void matrix_inv(double** matrix, int N){
   double ratio,a;
   for(int i = 0; i < N; i++){
      memset(matrix[i]+N,0,N*sizeof(double));
      matrix[i][N+i] = 1.0;
   }
   for(int i = 0; i < N; i++){
      for(int j = 0; j < N; j++){
         if(i!=j){
            ratio = matrix[j][i]/matrix[i][i];
            for(int k = 0; k < 2*N; k++){
               matrix[j][k] -= matrix[i][k]*ratio;
            }
         }
      }
   }
   for(int i = 0; i < N; i++){
      a = matrix[i][i];
      for(int j = 0; j < 2*N; j++){
         matrix[i][j]/=a;
      }
   }
}

//FILE* yu;
int M;
int d=0;
int N;
```

```
std::vector<double> yV,uV,tV;
//char syu[80];

double **X;
double **Xt;
double **T;
double **R;
double **S;
double *Y;
double **Q;
double *A;

timespec t1,t2;
float dt;

int main(int argc, char *argv[]){
   if(argc<3){
      printf("Использование $%s M N\n",argv[0]);
      printf("M-порядок, N-колич.точек\n");
      return 0;
   }
   M=atoi(argv[1]);
   N=atoi(argv[2]);

   X=new double*[N];
   Xt=new double*[2*M];
   Q=new double*[2*M];
   R=new double*[2*M];
   S=new double*[2*M];
   T=new double*[2*M];
   A=new double[2*M];
   Y=new double[N];

   for(int i=0;i<N;i++){
      X[i]=new double[2*M];
   }

   for(int j=0;j<2*M;j++){
      Xt[j]=new double[N];
      Q[j]=new double[N];
      R[j]=new double[2*M];
      S[j]=new double[4*M];
   }

   for(int i=0;i<N;i++)
      Y[i]=(rand()%5000000L-2500000L)/1000.0;//yV[i+M];

   for(int i=0;i<N;i++){
      for(int j=0;j<M;j++){
         X[i][j]=  (rand()%5000000L-2500000L)/1000.0;//(-1.0)*yV[i+M-j-1];
```

```
            Xt[j][i]=X[i][j];
        }
    }

    for(int i=0; i<N;i++){
        for(int j=0;j<M;j++){
            X[i][j+M]= (rand()%5000000L-2500000L)/1000.0;//uV[i+M-j-1];
            Xt[j+M][i]=X[i][j+M];
        }
    }


            clock_gettime(CLOCK_REALTIME, &t1);

            mat_mul(Xt,X,R,2*M,N,2*M);


            for(int i=0;i<2*M;i++){
                memcpy(S[i],R[i],2*M*sizeof(double));
            }

            matrix_inv(S,2*M);

            for(int i=0;i<2*M;i++){
                T[i]=S[i]+2*M;
            }

            mat_mul(T,Xt,Q,2*M,2*M,N);

            mat_mul_vec(Q,Y,A,2*M,N);

    clock_gettime(CLOCK_REALTIME, &t2);

            dt = (t2.tv_sec-t1.tv_sec)*1000000.0+(t2.tv_nsec-t1.tv_nsec)/1000.0;
            for(int i=0;i<2*M;i++){
                fprintf(stderr,"a[%d]=%f\n",i,A[i]);
            }
            fprintf(stderr,"t=%f\n",dt);
            printf("%d %d %f\n",M,N,dt);

    for(int i=0;i<N;i++){
        delete[] X[i];
    }

    for(int j=0;j<2*M;j++){
        delete[] Xt[j];
        delete[] Q[j];
        delete[] R[j];
        delete[] S[j];
    }
```

```
    delete[] X;
    delete[] Xt;
    delete[] Q;
    delete[] R;
    delete[] S;
    delete[] T;
    delete[] A;
    delete[] Y;

    return 0;
}
```

2. Выполнить оптимизацию путём распроцедуривания:

Дело в том, что в данном случае решается задача, которую можно рассматривать, как одно целое, поэтому нет смысла делить ее на подзадачи в соответствии с принципами процедурного программирования.

Т.е. ряд вызовов функций

```
mat_mul(Xt,X,R,2*M,N,2*M);

for(int i=0;i<2*M;i++){
   memcpy(S[i],R[i],2*M*sizeof(double));
}

matrix_inv(S,2*M);

for(int i=0;i<2*M;i++){
   T[i]=S[i]+2*M;
}

mat_mul(T,Xt,Q,2*M,2*M,N);

mat_mul_vec(Q,Y,A,2*M,N);
```

Можно заменить на один:

```
mnk(N, M, X, Xt, Y, A, S, Q);
```

При этом можно будет произвести дополнительную оптимизацию:

- промежуточные матрицы R и T станут не нужны;
- вместо промежуточной матрицы Q можно будет использовать вектор;
- в коде умножения матриц $X^T X$ можно будет учесть тот факт, что получается всегда симметричная матрица, и поэтому можно вычислять только элементы выше диагонали и саму диагональ;
- этапы вычисления можно «сцепить» так, чтобы внешний цикл предыдущего этапа начинал вычисления для следующего этапа.

3. Откомпилировать обе программу с ключом -O2 и выполнить сравнение затрат времени для следующих пар значений M и N:
   4 80
   8 160

16 320
24 480
40 800

Каждая из программ выдаёт 5 значений затрат времени. Из этих значений следует отбросить два крайних, явно выпадающих значения, а по оставшимся трём рассчитать среднее арифметическое.

4. Результаты свести в таблицу:

Таблица 1. Сравнение затрат времени.

| Размерность | Исходная | Оптимизиров. | OMP | Intrinsic | OMP+Intrinsic |
|---|---|---|---|---|---|
| 4x80 | | | | | |
| 8x160 | | | | | |
| 16x320 | | | | | |
| 24x480 | | | | | |
| 40x800 | | | | | |

5. Переработать программу из пункта 2 так, чтобы она включала в себя использование векторных операций путем применения intrinsic-функций. Откомпилировать программу с ключами -O2 и -mavx и выполнить для тех же пар значение M и N, что и в предыдущем пункте. Убедиться, что результаты расчётов совпадают. Полученные результаты занести в таблицу.
6. Переработать программу из пункта 2 так, чтобы она включала в себя использование OpenMP при помощи директив компилятору На старых процессорах многопоточность работает нестабильно, поэтому запуск программы необходимо осуществить несколько раз до получения приемлемого результата. Полученные данные занести в таблицу.
7. Добавить в программу из пункта 5 директивы компилятору, такие же как пункте 6. Также как и в пункет 6 осуществить многократный запуск программы для каждой размерности до получения приемлемых результатов. Полученные данные занести в таблицу.
8. По данным в таблице построить графики. Для этого рекомендуется использовать библиотеку matplotlib.
   Сделать выводы.

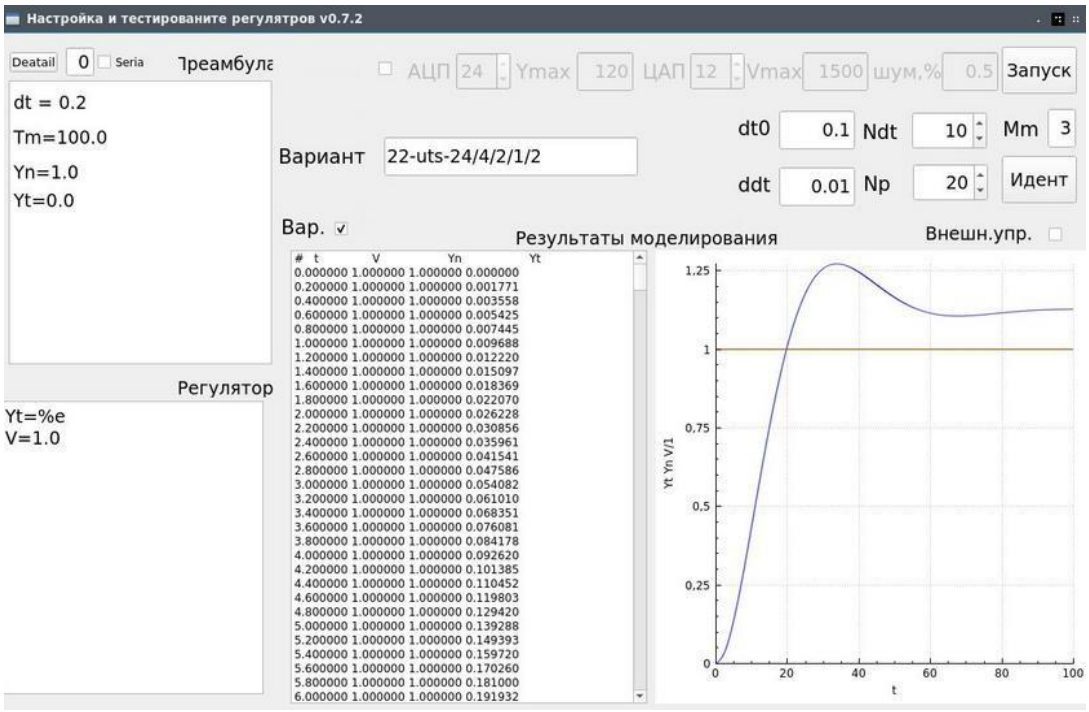
## Раздел II
## Использование полученного кода для решения конкретной задачи.

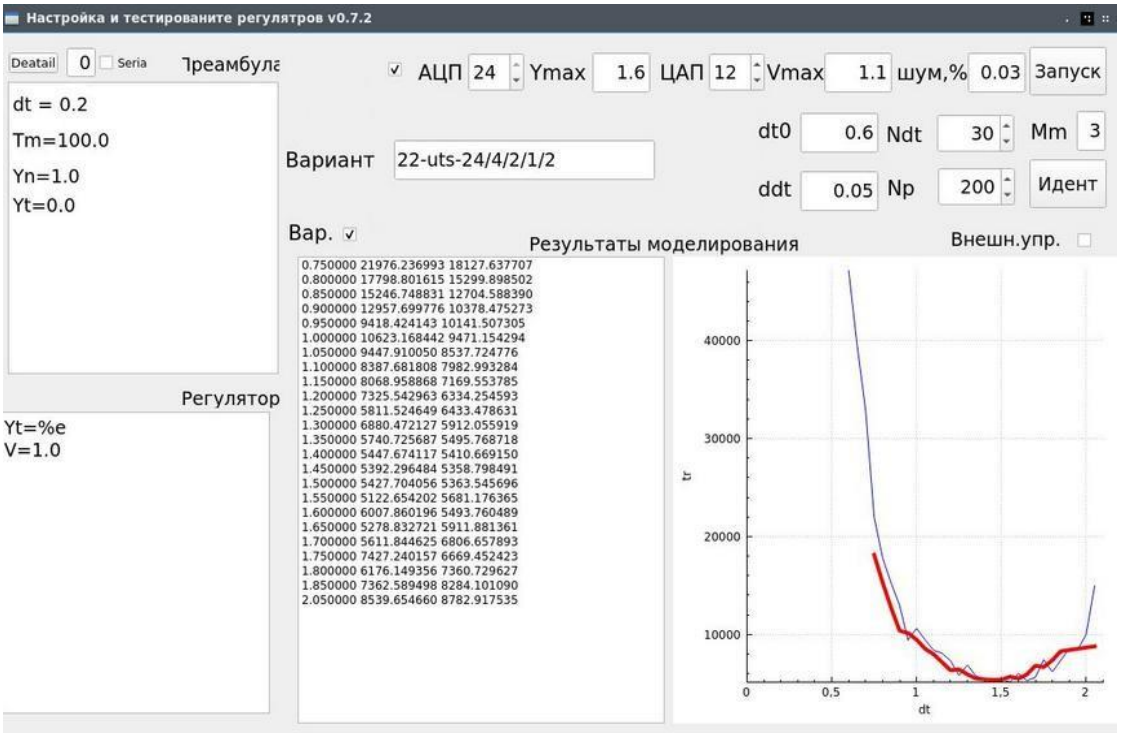
Снимаем переходную характеристику объекта, задав свой вариант и продолжительность разгона чуть больше, чем время выхода на установившееся значение. Шаг по времени должен быть таким, чтобы получилось не менее 200 точек.

Если дважды кликнуть левой кнопкой мыши по окну с данными, то откроется диалог сохранения файла, который можно назвать razgon.txt. В этот файл будет записано содержимое окна. Эти данные нам понадобятся для верификации модели объекта в виде разностной передаточной функции.

Для идентификации объекта строим зависимость следа ковариационной матрицы от кванта времени. Ожидаемый оптимум находится в окрестностях точки, равной 0.01 времени выхода объекта на установившееся значение, т.е. 1 сек. Задаём стартовый квант 0.6 сек и с шагом 0.05 сек снимаем 30 точек по 200 измерений на каждую. Не всегда выходит получить с первого раза подобный график. Поэтому приходится играть значениями.

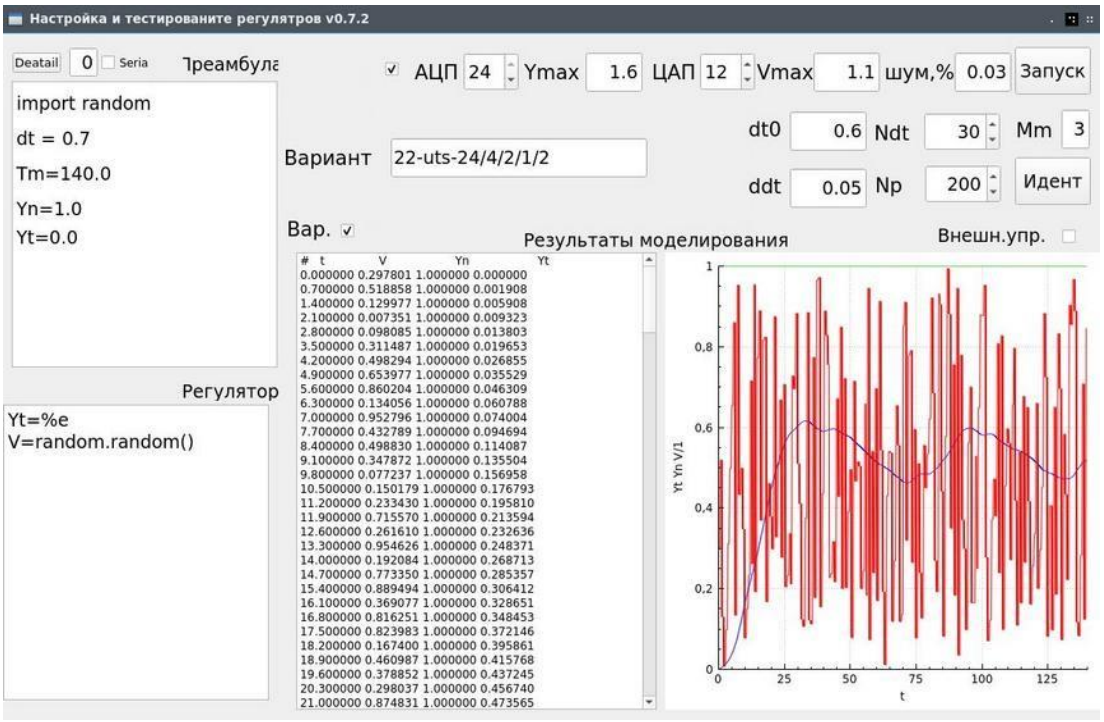
Также следует подключить эмуляторы АЦП на 24 разряда и ЦАП на 12 разрядов и задать шум 0.03%.




Таким образом оптимальный квант времени находится в окрестностях 1.4 сек.

Задаём оптимальный квант времени и подаём на вход данные от генератора случайных чисел. Tm следует выбрать таким, чтобы получилось около 200 точек.




Полученную кривую следует сохранить в файл ident.txt

Для идентификации параметров объекта управления требуется текстовый файл с тремя колонками: время, вход, выход. Т.е. из полученного файла надо вырезать колонку с нормированным значением. Для вырезания прямоугольного блока текста лучше использовать текстовый редактор Kate. Также следует удалить строчку с заголовком.

Порядок формирования матрицы X и вектора Y по результатам экспериментальных данных подробно изложен в соответствующих учебниках.

Можно воспользоваться следующим кодом, который получает в качестве параметров командной строки порядок модели объекта и наименование текстового файла с результатами наблюдений и трех колонок (время-вход-выход):

```
FILE* yu; //файл с данными наблюдения объекта
int M;    //порядок модели
int N;    //количество точек наблюдения

std::vector<double> yV,uV,tV;
char syu[80];

double **X;
double **Xt;//транспонированная матрица X
double *Y;

int main(int argc, char *argv[]){
    if(argc<2){
      printf("Использование: %s n tvy\n",argv[0]);
      printf("n-порядок,tvy-файл:время|вход|выход\n",%s);
      return 1;
    }
    M=atoi(argv[1]);

    double u,y,t;
```

```
    yu = fopen(argv[2],"r");//открываем файл
    while(fgets(syu,80,yu)){//считываем из него данные
      sscanf(syu,"%lf %lf %lf",&t,&u,&y);
      yV.push_back(y);
      uV.push_back(u);
      tV.push_back(t);
    }
    N=yV.size()-M; //определяем размер задачи
    N=(N/4)*4; //программа учебная и размер обязательно кратен 4

//Выделяем память под матрицы и векторы

    for(int i=0;i<N;i++){
      X[i]= new double[2*M];
    }
    for(int j=0;j<2*M;j++){
      Xt[j]=new double[N];
    }
    //заполняем матрицу X и вектор Y исходными данными
    for(int i=0;i<N;i++)
      Y[i]=yV[i+M];

    for(int i=0;i<N;i++){
      for(int j=0;j<M;j++){
          X[i][j]=(-1.0)*yV[i+M-j-1];
          Xt[j][i]=X[i][j];
      }
    }

    for(int i=0; i<N;i++){
      for(int j=0;j<M;j++){
          X[i][j+M]=uV[i+M-j-1];
          Xt[j+M][i]=X[i][j+M];
      }
    }

выполняем расчёты
.............................................

выводим результаты


освобождаем память
    return 0;
}
```

Назовём полученный исходный файл mnk_optim.cpp и откомпилируем его.

Запустим полученный исполняемый файл и передадим ему в качестве параметров размерность модели объекта и файл с экспериментальными данными ident.txt

```
$./mnk_optim 4 ident.txt
-1.613824
0.202827
0.564447
-0.145970
0.006405
-0.001692
```

```
0.000689
0.003045
```

Для проверки корректности расчетов используем эталонную программу, основанную на научной библиотеке для языка C:

```
$./mnk_s64 4 ident.txt
0.700000
139.300000
4
-1.613824
0.202827
0.564447
-0.145970
0.006405
-0.001692
0.000689
0.003045
```

Как видно данные совпадают. Такое совпадение бывает не всегда, поэтому вполне допустимы незначительные расхождения.

Вывод данной программы содержит шаг по времени, общее время, и порядок системы. Если эти данные подать на вход программы get_model_64, то она сгенерирует кривую разгона объекта по его разностной передаточной функции.

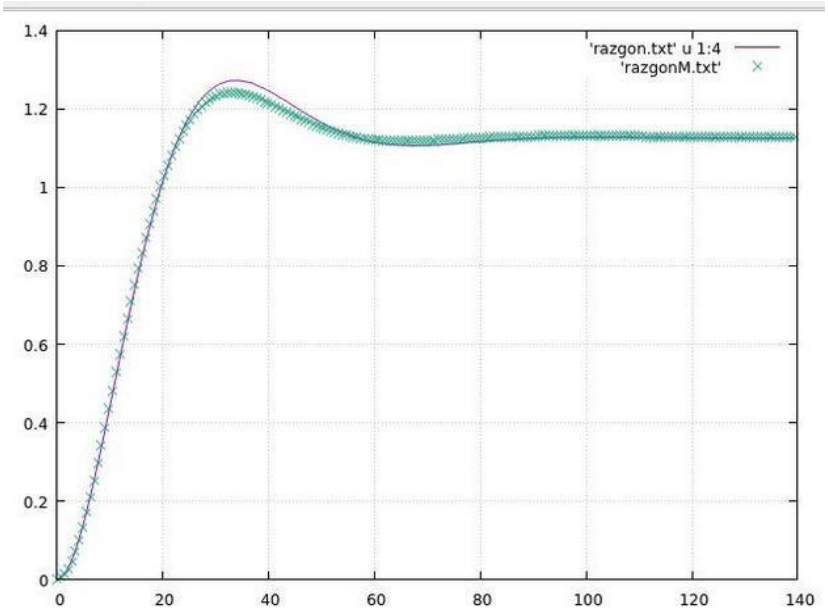
Воспользуемся этим для верификации полученной модели, а именно: запишем кривую разгона разностной модели в файл:

```
$ ./mnk_optim 4 ident.txt | ./get_model_64 > razgonM.txt
```

А затем в одних координатных осях построим графики разгона объекта полученные экспериментально и по его разностной передаточной функции:

Для этого используем программу gnuplot:

```
plot 'razgon.txt' u 1:4 w l, 'razgonM.txt
```



Как видно, совпадение достаточно хорошее. Поэтому полученная разностная передаточная функция может быть использована для построения цифрового регулятора.

## Раздел III
## Проектирование вычислительной станции.

Выбрать конструктор вычислительной станции в соответствии со своим вариантом:

1. https://www.team.ru/comp/workstation-builder.php?model=C621
2. https://www.team.ru/server/supermicro_builder.php?model=W4-E52
3. https://www.team.ru/comp/workstation-builder.php?model=WRX80
4. https://www.team.ru/comp/workstation-builder.php?model=X570
5. https://www.team.ru/comp/workstation-builder.php?model=C422
6. https://www.team.ru/comp/workstation-builder.php?model=X299
7. https://www.team.ru/comp/workstation-builder.php?model=C242
8. https://www.team.ru/comp/workstation-builder.php?model=C246
9. https://www.team.ru/comp/workstation-builder.php?model=W480

Выбрать марку и количество процессоров тип и количество графических ускорителей, а также объем ОЗУ по максимуму.

Поместить спецификацию на полученную вычислительную станцию в Таблицу 2.

Таблица 2. Спецификация на вычислительную станцию

| № | Наименование | Цена | Колич. | Стоим. | Пик.производит. |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| Итого | | | | | |

На основе данных, представленных на https://www.team.ru/lab/xeon_phi_test.shtml#a и https://www.team.ru/lab/nvidia_tesla_new.shtml, рассчитать пиковую производительность спроектированной рабочей станции.

В отчёт включить листинг последнего варианта программы спояснениями и комментариями, все графики, а также спецификацию на рабочую станцию.