



Пакеты научных вычислений

Лекция 4. Процедуры и элементы программирования

Курбатова Наталья Викторовна, к.ф.-м.н.,
доцент кафедры математического
моделирования, мехмат, ЮФУ



Содержание:

- Процедуры **Inline**
- **Anonymous**. Непоименованные процедуры
- Процедуры с параметрами и без параметров
- Подпроцедуры
- Процедуры с переменным числом I/O параметров
- **Feval**. ВПримеры
- Операторы программирования: условный, цикла, переключатель, перехвата исключительных ситуаций



СИСТЕМНЫЙ КОНСТРУКТОР INLINE

`Name1=inline(funChar);` % Name1 – inline, funChar –
выражение строкой

`Args=symvar(funChar);` % Args – содержит аргументы funChar

`g = inline(funChar, arg1, arg2,...)`

`isa(g, 'inline')` % контроль типа

INLINE не используется в старших версиях!
Вместо inline -- anonymous functions!

```
1 %% inline
2 z=inline('x.^2+y.^2-1');
3 z(2,3)
4 arg=symvar(z)
```

arg = 2x1 **cell** array

'x'

'y'

z: 1x1 inline =

Inline function:

val(x,y) = x.^2+y.^2-1



Anonymous function

FuncHandle = @FunctionName;

@ - конструктор function_handle

FuncHandle = @(ArgList)ExpressionChar

myAnonymous.m

```
function myAnonymous % без параметров
```

```
h=@(x)x.^2.*sin(x); % anonymous
```

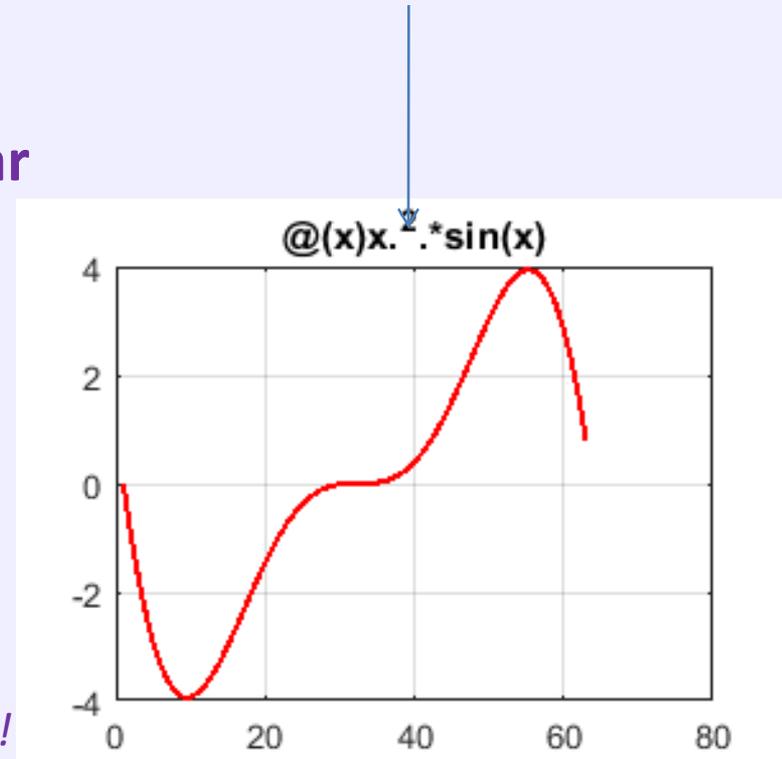
```
val=-pi:0.1:pi; % h(val) – вычисляет вектор ans
```

```
FuncPlot(h, val)
```

```
function FuncPlot(h, val) % подпроцедура
```

```
if isa(h, 'function_handle') % Check h  
                                as a function_handle!
```

```
A = h(val); % Construct A as h-function of val  
plot(A,'r-', 'linewidth',1.5); % Plot the resulting data  
title(func2str(h)), grid on % Convert function to Char  
end
```





Процедуры без параметров (ПБП)

Пример: файл **myFunction.m**

```
function myAnonymous % без параметров
```

```
h=@(x)x.^2.*sin(x); % anonymous
```

```
val=-pi:0.1:pi;
```

```
h(val) % вычисляет вектор ans - (непоименованная - Anonymous)
```

- ПБП - альтернатива script-файлу *.m
- Все переменные локальные
- Исполнение быстрее (нет операций I/O)
- Имя script-файла и процедуры совпадают

П Р Е И М У Щ

**Е
С
Т
В
А**



Подпроцедуры

- Подпроцедуры – находятся в теле головного файла
- Подпроцедура может иметь в т.ч. имя системной процедуры; приоритет выполнения у подпроцедуры
- Подпроцедура «не видна» любой другой программе (это позволяет f.e. переопределять стандартные функции – потрафить чувству собственной значимости)
- Подпроцедура выполняется быстрее внешней процедуры

Пример подпроцедуры `FuncPlot(h, val)` см. слайд 4

Заметим, что подпроцедура, например, `FuncPlot(h, val)` может не иметь выходных параметров!



Функции

- У функции **один** выходной параметр («условно» один)!
- Функция может быть частью выражения, **в отличие от?**
- Имя функции должно совпадать с именем файла, в котором она определена!
- Последний исполняемый оператор должен присваиваться идентификатору с именем функции!
- Функцию определяет синтаксическая конструкция:

NameF.m % имя файла

Шапка функции:

```
function outputmy=NameF(input1,...,inputm) % типы аргументов явно не указываются!
```

```
% 1) описание назначения параметров
```

Тело функции: {её основное содержание}

```
% последний исполняемый оператор:
```

```
outputmy=expression
```

```
end % не обязательно
```



Процедуры

- Процедура **не** может быть частью выражения!
- Имя процедуры должно совпадать с именем файла, в котором она определена!
- Имя внешней процедуры не может совпадать с именем встроенной (builtin) процедуры (функции)!
- На количество параметров I/O не накладывается ограничений!
- Процедуру определяет та же синтаксическая конструкция, что и функцию:

Шапка процедуры:

```
function [out1,...,outk]=NameF(input1,...,inputm) %
```

типы явно не указываются!

% описание типов и назначения параметров

Тело процедуры: {назначение - определить
out1,...,outk }

end % не обязательно



Процедуры с переменным числом I/O параметров

Шапка процедуры:

```
function [out1, ..., outk, varargout]=NameF(inp1, ..., inpn, varargin)
```

% назначении параметров:

% inp1, ..., inpn – обязательные входные параметры

% varargin – **системный** массив ячеек, содержит входные параметры, которые пользователь передал процедуре во время обращения, помимо обязательных;

% out1, ..., outk – обязательные выходные параметры

% varargout – **системный** массив ячеек, содержит выходные параметры, необходимые пользователю, помимо обязательных;

Программист в назначении параметров (help внутри процедуры) указывает какие входные/выходные (I/O) параметры предусмотрены в данной процедуре!

Тело процедуры (определение I/O параметров):

p=nargin % определяется *количество всех входных параметров*, тогда

p-n % – количество входных параметров в **varargin** при текущем обращении

m=nargout % определяется *количество всех выходных параметров*, тогда

m-k % – количество входных параметров в **varargout** при текущем обращении



Процедуры с переменным числом I/O параметров (продолжение)

Тело процедуры:

1) определение размера массивов I/O параметров переменной длины:

`p=nargin; m=nargout`

`pn=p-n, mk=m-k % размер varargin и varargout, соответственно`

2) Присвоение локальным переменным значений из varargin:

`if (p-n>0), Par_1=varargin{1}`

`elseif (p-n>1)`

`Par_2=varargin{2}`

`ooo`

`elseif(p-n>p-n-1) % or else`

`Par_pn=varargin{p-n}`

`end`

3) Основной АЛГОРИТМ

4) Формирование varargout:

`if (m-k>0)`

`varargout{1}=expression_1;`

`elseif (m-k>1)`

`varargout{2}=expression_2;`

`ooo`

`elseif(m-k>m-k-1) % or else`

`varargout{m-k}=expression_mk;`

`end`

Вместо *условного оператора* можно использовать переключатель **switch, case, otherwise, end;**



Выполнить функцию. Feval

СИНТАКСИС:

$[y_1, \dots, y_n] = \text{feval}(\text{fun}, x_1, \dots, x_k)$ – выполнить функцию (x_1, \dots, x_k) – её аргументы (один или несколько); fun – `function_handle` **или** имя встроенной функции **или** внешняя функция.

Пример использования:

Головная процедура без параметров:

```
n=3, for i=1:n
    namef={'cos', '@sin', 'myextFun'};
    result(i)=feval(namef{i}, pi/6);
end
[result(1:n)]
```

Внешний файл `myextFun.m`

```
function r=myextFun(arg)
r=(sin(arg))^2
End
```

Имя файла и имя внешней функции совпадают!

Объясните чем являются элементы `namef`?

Проверьте `>>which('cos')`



Условный оператор

короткая форма: `if expression, statements, end;`

полная форма: `if expression1`

`statements1`

`elseif expression2`

`statements2`

`else`

`statements3`

`end`

expressions: `checktype`(see `help is*` – `isempty`, `isfinite`, `isnumeric`, `isvarname` . . .),

relational operators (**help relop**),

logical (`all`, `any`), `exist`(string)

statements: последовательность команд – основной код



Синтаксические аналоги циклов в ML

Упрощение синтаксиса в ML (замещение циклов):

- Поэлементное присвоение B , $A(i,j) = B(i,j) \leftrightarrow A=B$
- Присвоение матрице или её части константы, A - \exists :
 $[m,n]=\text{size}(A)$, $A(1:k,p:p+k-1)=pi$ или $A(:,:)=pi$ (A так? $A=pi$)
- Приёмы оптимизации: f.e. суммирование элементов вектора: b – вектор, $b*(\text{ones}(\text{size}(b)))'$
- Присвоение по условию (цикл + условие); ops: $=$, $>$, $<$, \sim ; $A(A \text{ ops value})=\text{expr}$; f.e. $(A(A>0)=\text{rand}(1))$



Цикл с известным и нет количеством повторений

Синтаксис

случай известного количества повторений:

короткая форма: **for** variable = expr, statements, **end**; % в строку

полная форма 1: **for** variable = initial_value:step:end_value
statements
end

полная форма 2: **for** variable = [vector] % or variable = [**matrix** or cell] ?
statements
end

initial_value – начальное значение индексной переменной (и.п.);

step – шаг изменения индекса; в случае **for variable = initval:endval** (step=1 – умолчанию);

end_value– конечное значение индексной переменной.

случай неизвестного количества повторений:

короткая форма: **while**, expression, statements, **end**;

полная форма: while expression
statements
end;



Примеры циклов:

- 1) for e=eye(n), e, end; % на i -м шаге вычисляется i -й вектор-столбец $e(:, i)$
- 2) while (A&B);
- 3) while (A|B);
- 4) while (b~=0) &(a/b>3*pi)
if exist('myfun.m')&(myfun(x)>=y)
if iscell(A) & all(cellfun('isreal',A)) % **объясните!** или см. детали >>help cellfun(fun,A)
- 5) eps=1; while(abs(1+eps)>1), eps=eps/2, end; eps=2*eps

Объяснения (2-5):

(2): если A – нулевая (лог.), то B – не вычисляется, результат для любого B – ложный;

(3): если элементы A – лог.единицы, матрица B – не вычисляется, результат для любого B – истина;

(4): цикл выполняется пока $a/b > 3\pi$ и **исключается** деление на ноль; **контролируется** существование и величина функции, вещественность элементов массива ячеек

(5): на первый взгляд – бесконечный цикл, однако он прекратится по достижении **машинного нуля**; **Хорошо бы выяснить количество шагов (делений)!**



Пример 1 бесконечного цикла

```
1 - clear all;
2 - file1 = fopen('output_code.txt','r'); % открываем output_code.txt
3 - %                                     для чтения
4 - file2 = fopen('line.txt', 'w');      % открываем line.txt
5 - %                                     для записи
6 - while 1 % выполняется всегда
7 - tline=fgetl(file1);
8 - if ~ischar(tline), break, end        % выход из цикла по break
9 - fprintf(file2, '%s;\n', tline);     % подавляем строку-вывода ";"
10 - disp(tline)
11 - end
12 - fclose(file1);
13 - fclose(file2);
```

edit_string.m × input_code

**fgetl (file) - % чтение
строк (lines) текст.файла**

status = feof (fileID) возвращает 1,
если предыдущая операция "чтения"
устанавливает индикатор конца файла

```
1 clear
2 x=0:0.1:pi
3 y=sin(x)
4 plot(x,y)
5 hold on
6 grid on
7 g=cos(x)
8 plot(x,g)
9 title('two graphs')
10 disp('thats all!')
```

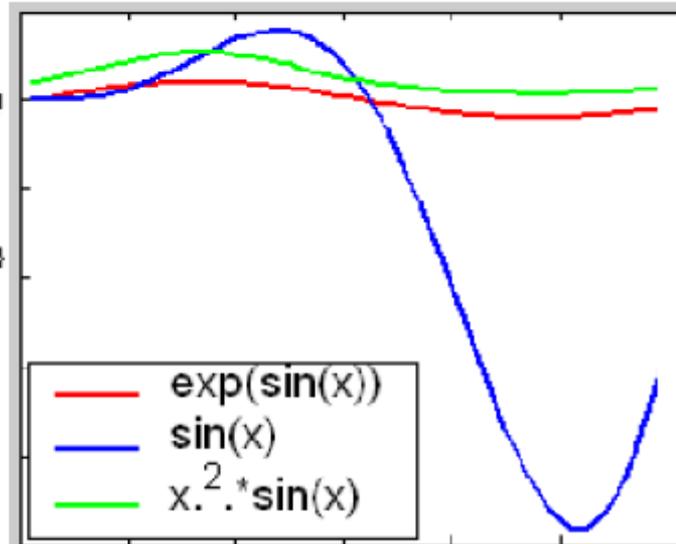
```
1 clear;
2 x=0:0.1:pi;
3 y=sin(x);
4 plot(x,y);
5 hold on;
6 grid on;
7 g=cos(x);
8 plot(x,g);
9 title('two graphs');
10 disp('thats all!');
```

edit_string.m × input_code × output_code.txt ×



Пример 2 бесконечного цикла

```
1 - clear
2 - % example if
3 - a=randi(10)
4 - if nnz(a)>prod(size(a))/2
5 -     s=sparse(a)
6 - end
7 - % example for
8 - x=[0:0.1:2*pi]';
9 - mycolor={'red','blue','green'}
10 - leg={}
11 - A={'sin(x)', 'x.^2.*sin(x)', 'exp(sin(x))'}
12 - for f=A
13 -     k=1
14 -     while ~strcmp(A{k},f{:})
15 -         k=k+1
16 -     end
17 -     r=plot(eval(f{:})),
18 -     set(r,'color',mycolor{k})
19 -     set(r,'linewidth',1.5)
20 -     leg=union(leg,f)
21 -     hold on
22 - end
23 - t=legend(leg{:})
24 - set(t,'fontsize',14)
```





Переключатель

```
switch calculated _expression  
case possible _expression_1  
statements,  
...  
case possible _expression_end  
statements,  
end
```

calculated _expression – вычисляемое выражение, в т.ч. строка;
сравнивается со значениями

possible _expression_1, ... possible _expression_end;

при совпадении выполняются инструкции “помеченной” траектории

Пример переключателя:

```
method = 'Bilinear';  
switch method  
case {'linear', 'bilinear'}  
disp('Method is linear')  
case 'cubic'  
disp('Method is cubic')  
otherwise  
disp('Unknown method')  
end
```



Оператор перехвата

```
try  
statements  
catch  
statements  
end
```

```
>> a=1:10; a*a  
Error using * Inner matrix  
dimensions must agree.  
  
>> lasterr  
ans =  
Error using * Inner matrix  
dimensions must agree.
```

- Системная переменная *lasterr* хранит информацию о последнем сообщении об ошибке.
- На этапе *try – catch* возможна ошибочная ситуация.
- На этапе *catch – end* происходит обработка ошибки.
- Предусмотреть некоторый логически обусловленный список ошибок.
- С помощью оператора *findstr* определить суть ошибки.
- С помощью переключателя *switch* выбрать ветвь алгоритма.



Бонус

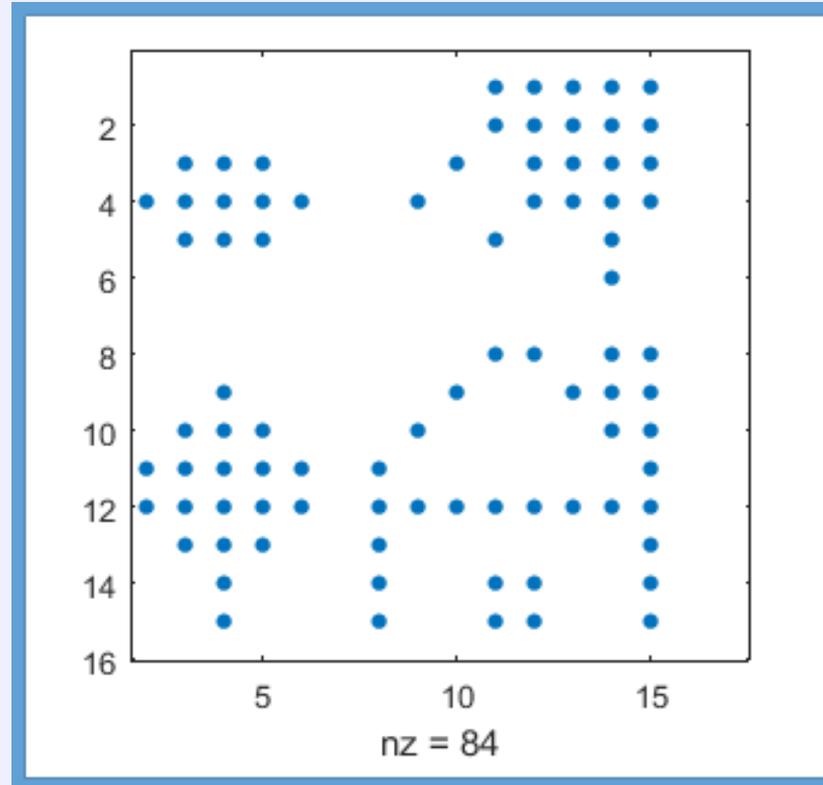
*В последних версиях MatLab “трудятся” функции
определения локальных минимумов и
максимумов:*

islocalmin и islocalmax !

*Интересно (?) заглянуть в код этих системных
функций, сравнить с алгоритмом в л.3 с.16 !!!*

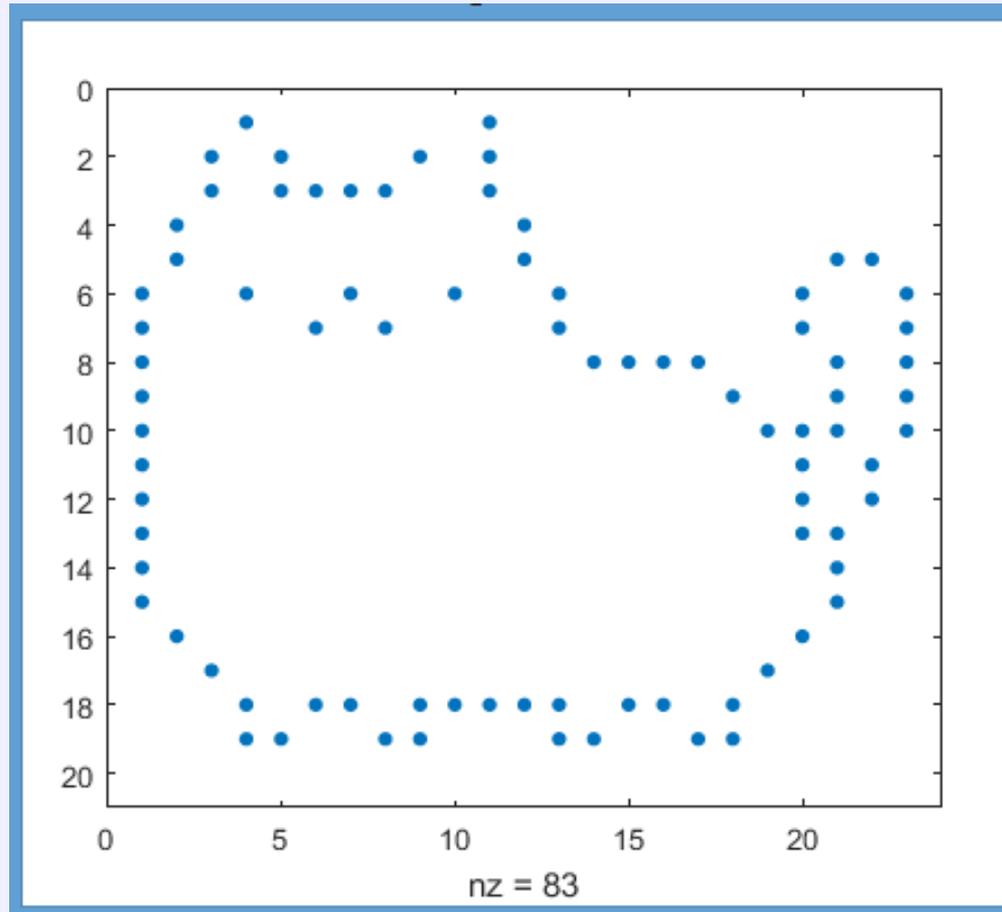


Вязников Илья - самый первый сделал, на паре! Первая группа



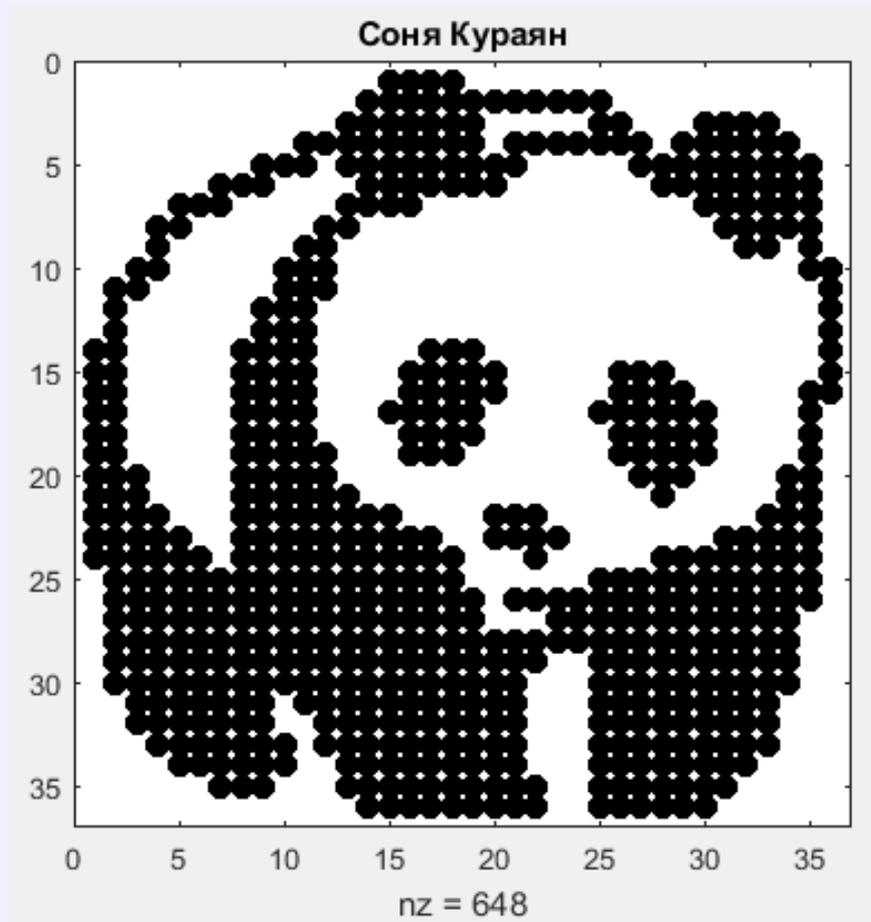


Тинчурын Руслан, группа 1



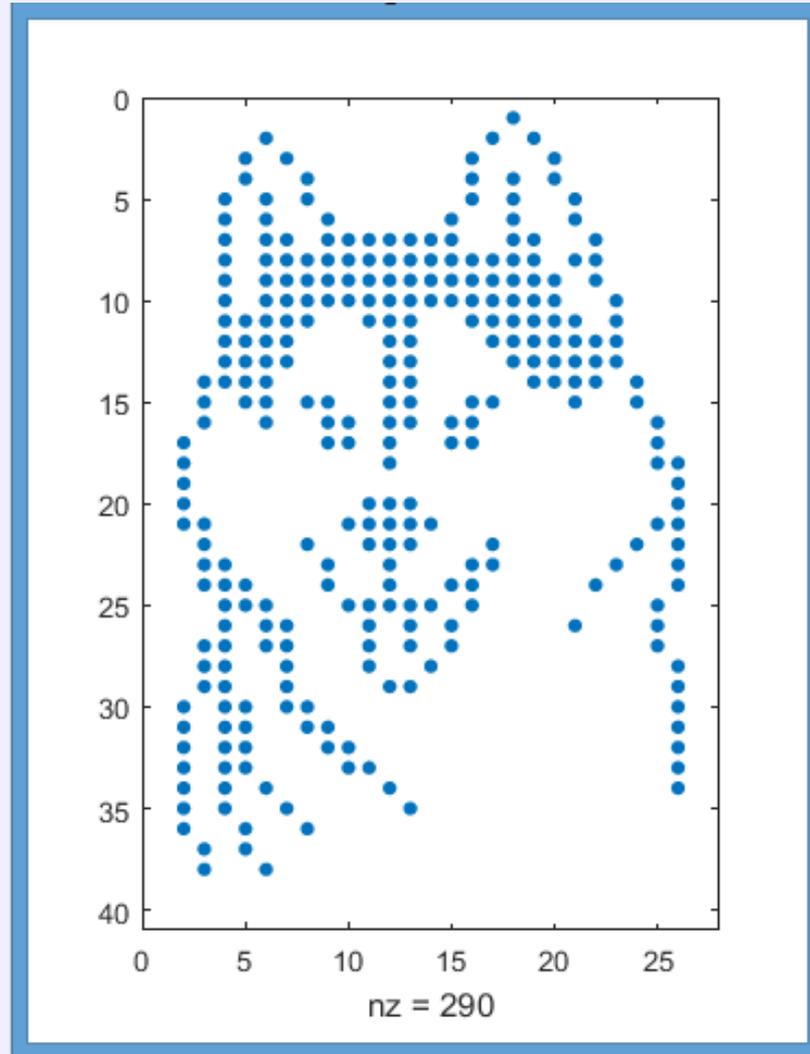


Софья Кураян, 4 группа



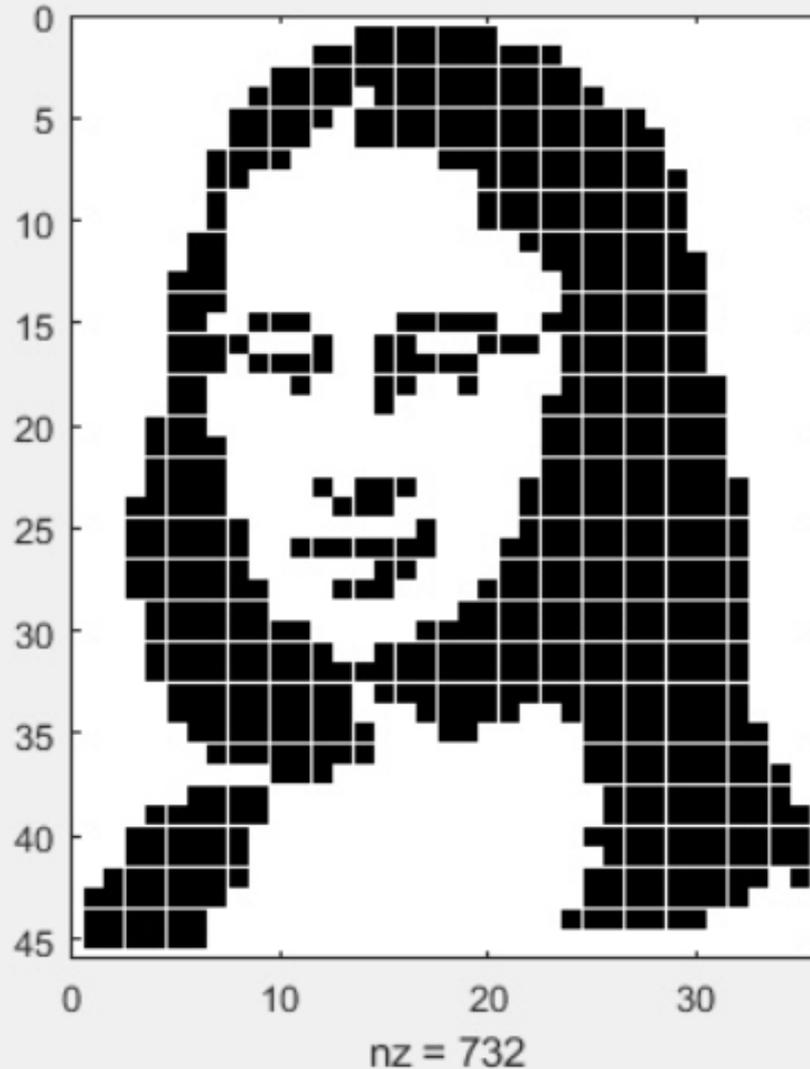


Мелехова Дарья, 1 группа



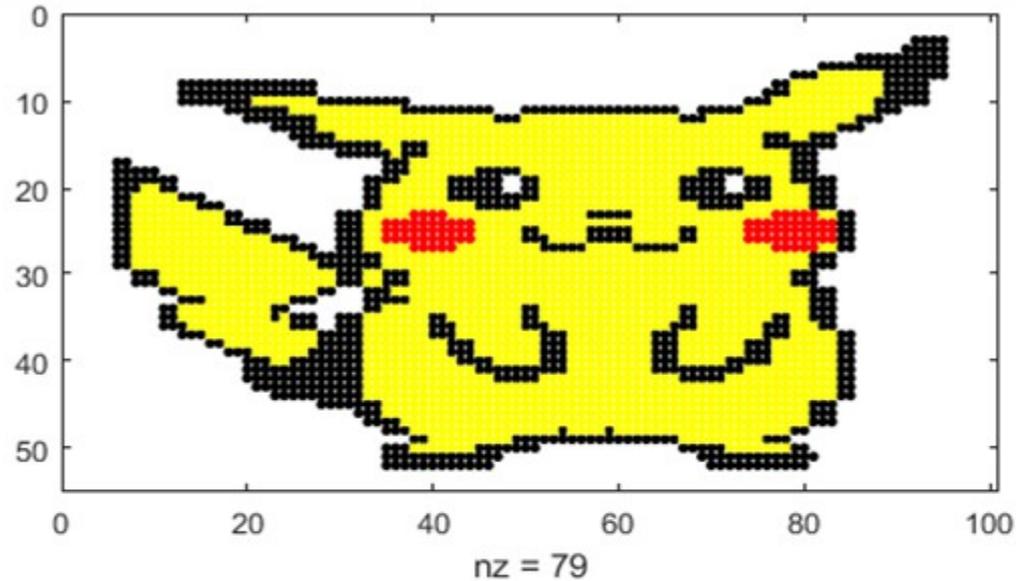


Захаренко Анастасия, Семенченко Артем
Турыгин Саша, Цулая Лена, 4 группа





Мовчан Максим, 4 гр.



```
spy(A == 1, 'black', 10)
hold on
spy(A == 2, 'yellow', 10)
spy(A == 3, 'red', 10)
hold off
```



2024

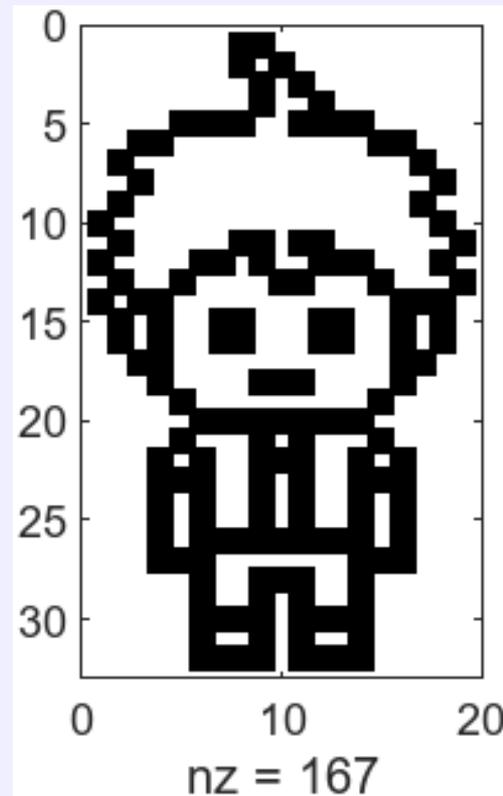


Рисунок Каплунова Тимура



2024

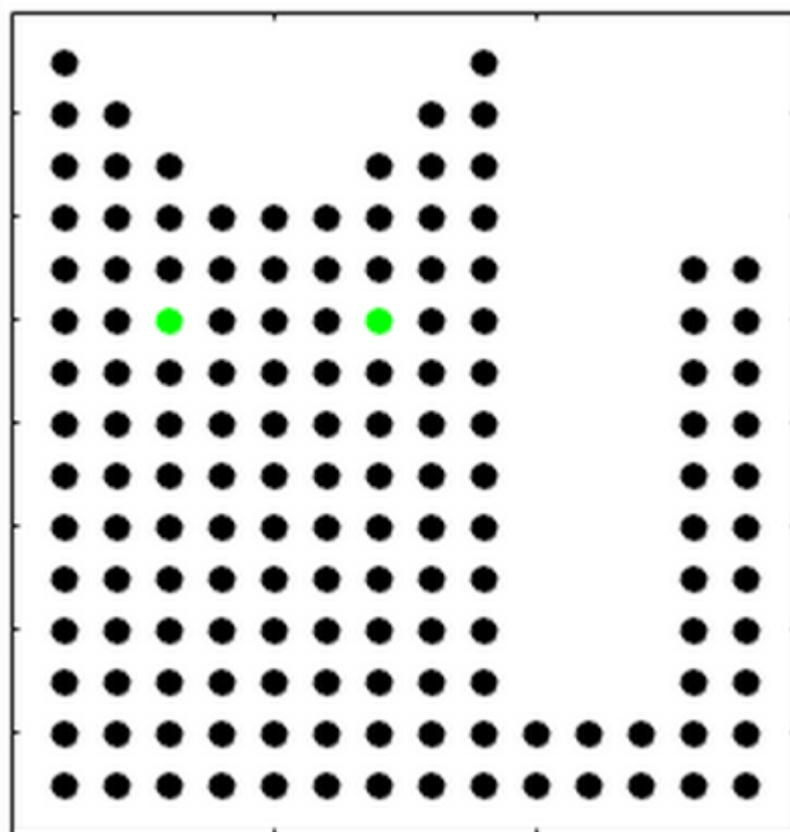


Рисунок Шпаковой Вероники



Спасибо за внимание!

“Глаза боятся, а руки делают”

Русская народная пословица!