MATLAB Programming Tricks

Retreat 2015 Bad Überkingen

Dominic Mai

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Why use MATLAB?

UNI FREIBURG

- Easy to write code
  - No type declarations needed
  - Memory management handled automatically
  - Structs for easy data keeping
  - Duck typing
- Built in nd-Arrays
  - Matrix-Vector arithmetic, linear algebra
- Lots of stuff available
  - Toolboxes (40+)
  - Matlabcentral
  - Interfaces to many 3rd party tools written in other languages: ANN, libSVM, LBFGS, Caffe,LSA TR, openCV, ...

Why use MATLAB?

UNI FREIBURG

- Easy to debug
  - Interpreted language with interactive workspace
- MEX C/C++ interface
- Profiler
- Easy to generate nice looking plots
- Well documented

Downsides?

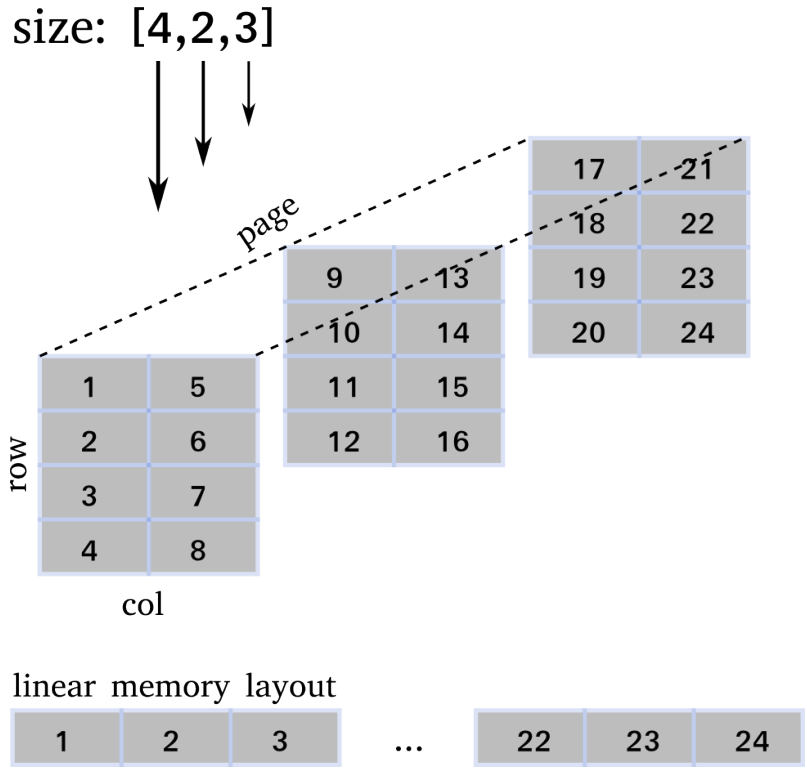
- Not free
- Interpreted language
  - Function calls and loops are (sometimes) relatively slow
- Vectorization (getting rid of loops)
  - Often not straight forward
  - Probably needs more memory
- Passing arguments to functions: Call by value
  - It's actually copy on write
  - there are ways around (→ mex, global)

Overview

- **Array Memory Layout**
- Image Transformations
- Random Stuff

ND Arrays: memory arrangement

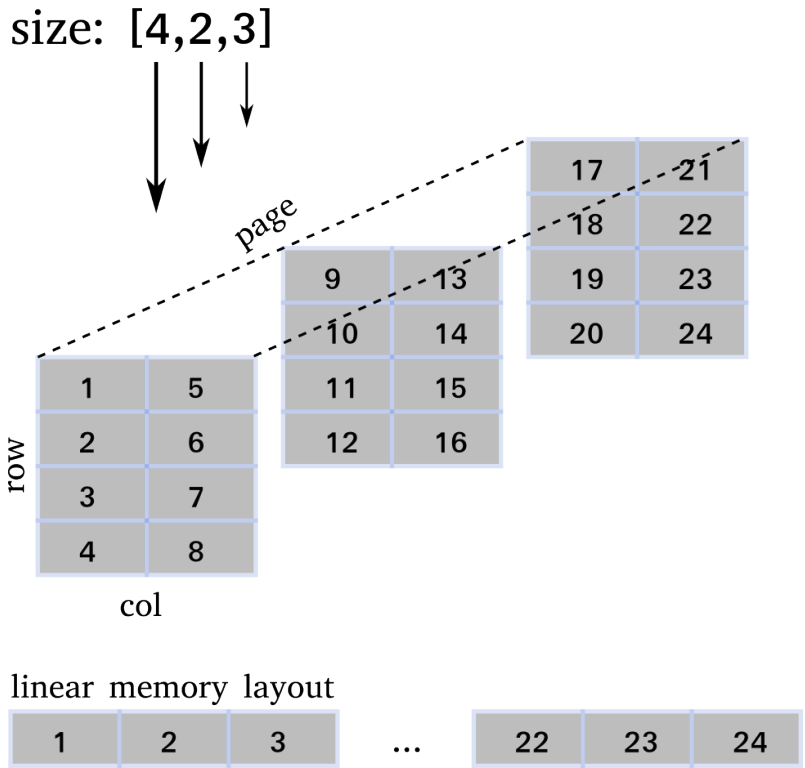
- MATLAB supports nd-arrays of arbitrary dimensions
  - Need continuous chunk of memory
  - Stored column major



ND Arrays: creationism

- Nested loops

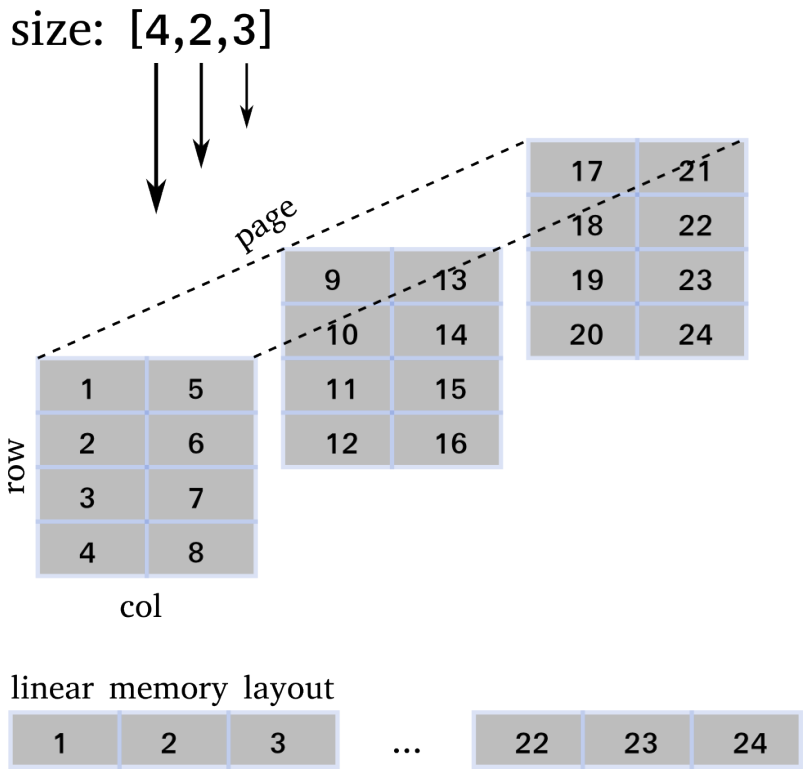
```
%fill up with nested for loop
c = 1;
for i3 = 1:3
      for i2 = 1:2
            for i1 = 1:4
                  A(i1,i2,i3) = c;
                  c=c+1;
            end
      end
end
```



ND Array - creationism

- Linear element access

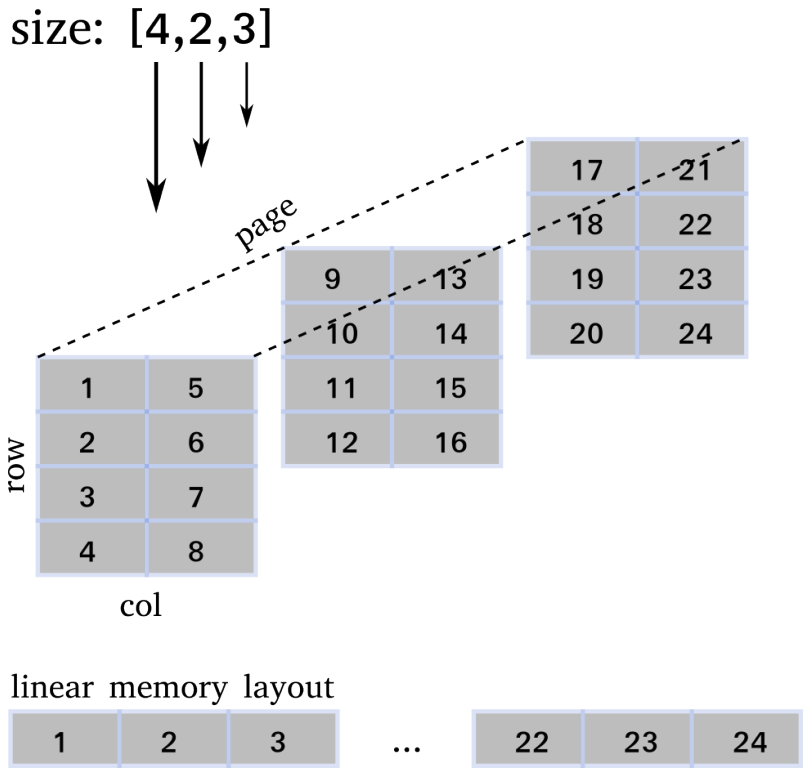
```
%linear element access
A = zeros([4,2,3]);%allocate & def. shape
for i = 1:numel(A)
      A(i) = i;
end
```



ND Array - creationism

- Concatenation

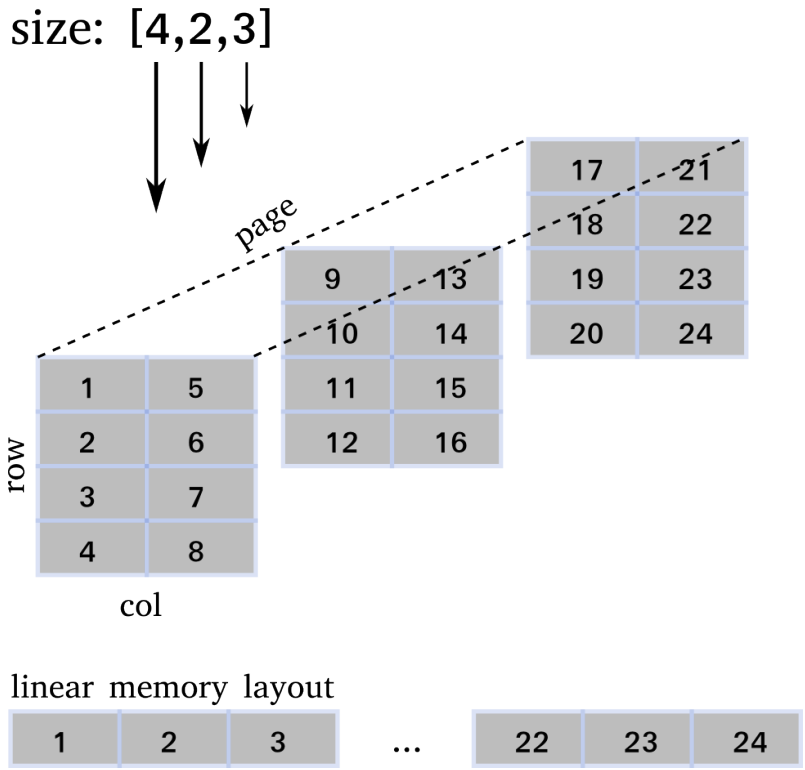
```
p1 = [1 2 3 4; 5 6 7 8]';
p2 = [9:12; 13:16]';
p3 = [17 21;18 22;19 23;20 24];

A = cat(3,p1,p2,p3);
```



ND Array - creationism

- reshape
  - Fills along fastest moving dimension

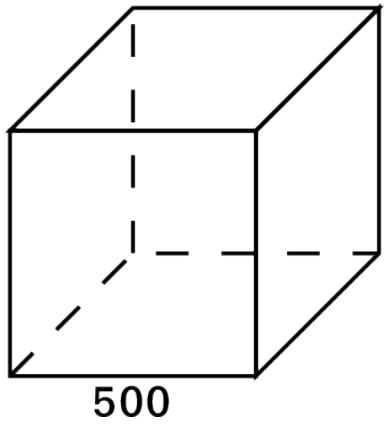
```
clear A;
A = reshape(1:24,[4,2,3]);
```



Nd Array element access

- 500^3 elements: sum up
  - Different nested loops
  - 3 dimensions: how many ways are there?

```
acc = 0;
t = tic;
for i3 = 1:N
      for i2 = 1:N
            for i1 = 1:N
                  acc = acc + A(i1,i2,i3);
            end
      end
end
d = toc(t);
```



Nd Array element access

- Linear access

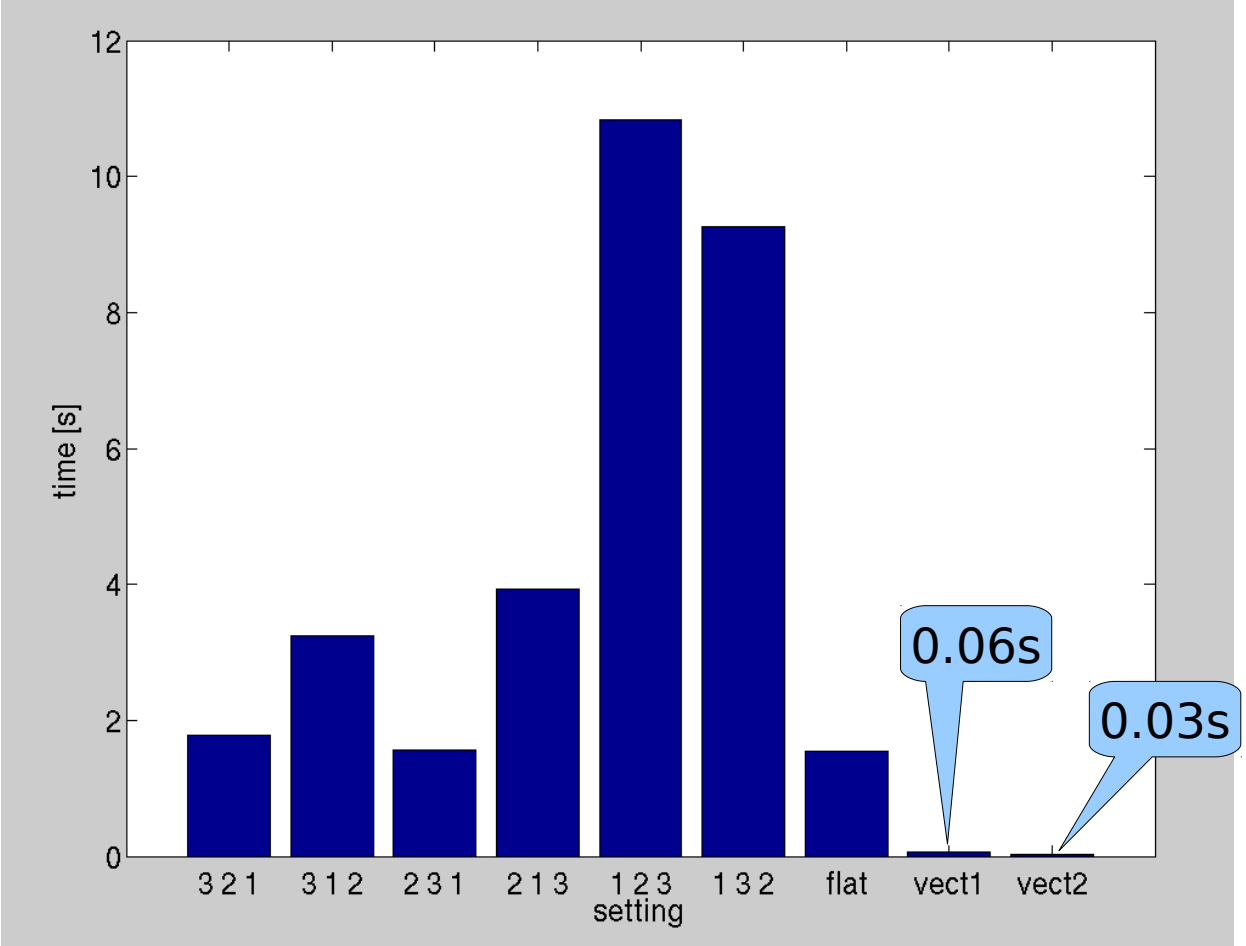
```
t = tic;
for i = 1:numel(A)
      acc= acc+A(i);
end
d = toc(t);
```

- Vectorized (i.e. no loops)

```
%v1: on flattened array
acc = 0;
t = tic;
acc= sum(A(:));
d = toc(t);

%v2: every sum reduces dim by 1
t = tic;
acc= sum(sum(sum(A)));
d = toc(t);
```

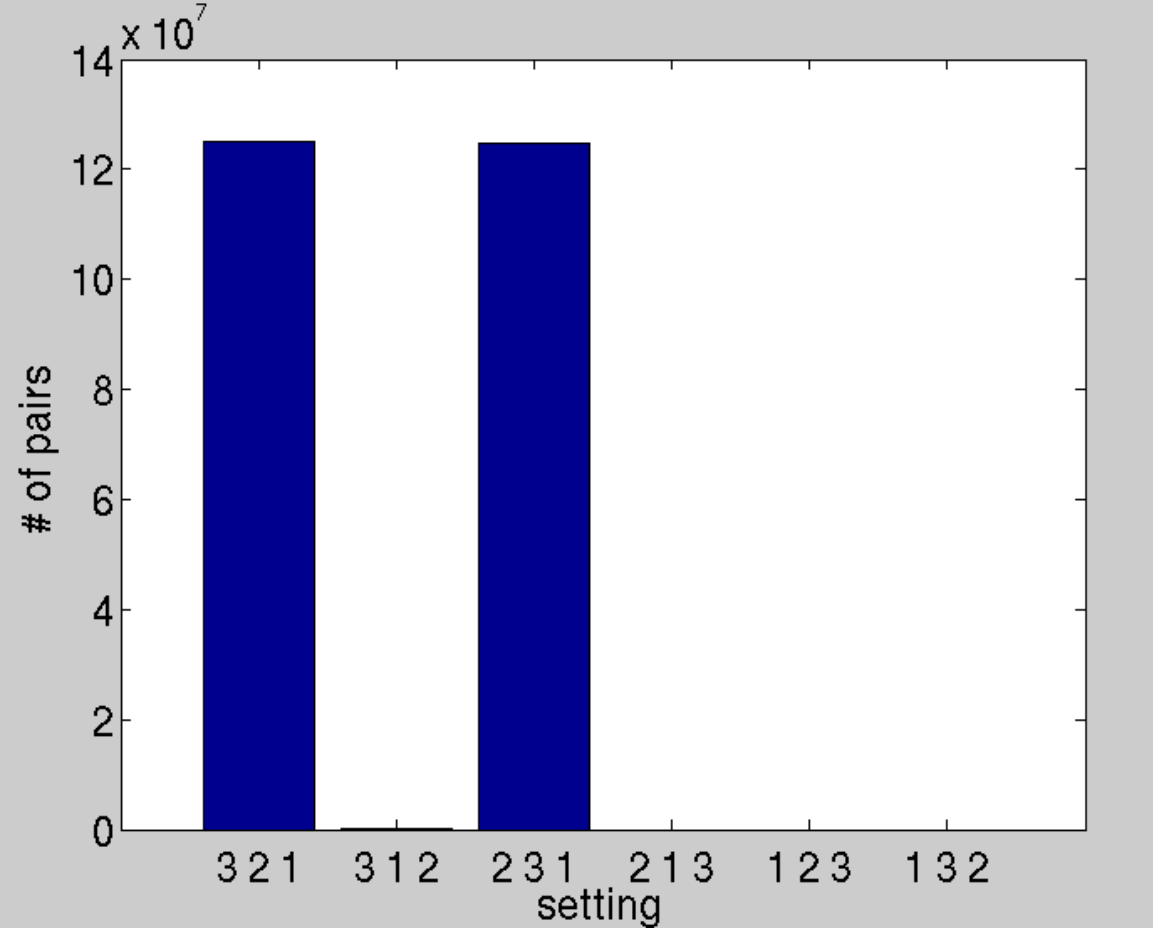
Nd Array element access

- Different nested loops, 500^3 elements



Nd Array element access

- Number of adjacent elements in memory



Nd Array element access: eval

- Create parameterizable code with **eval**

```
str= [  't=tic;',...
       'acc=0; ',...
       'for i%i=1:N ',...
             'for i%i=1:N ',...
                   'for i%i=1:N ',...
                         'acc=acc+A(i1,i2,i3); ',...
                   'end; ',...
             'end; ',...
       'end; ',...
       'd = toc(t);'];
```

Nd Array element access: eval

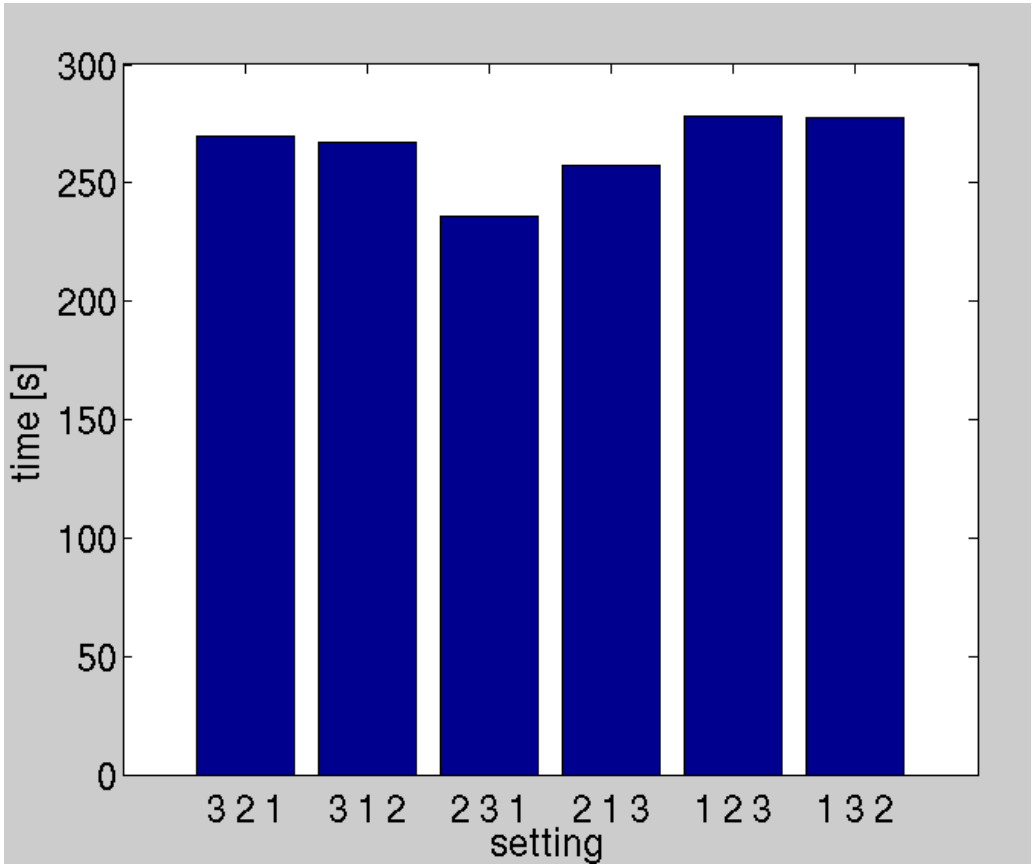
- Create parameterizable code

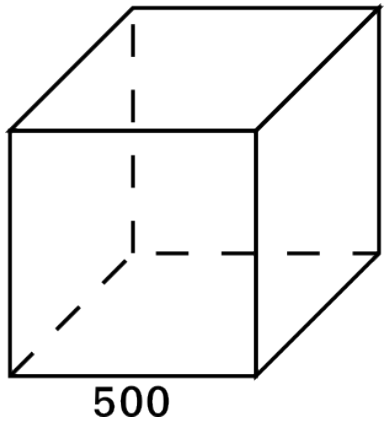
```
p = perms([1 2 3]);
for i = 1:size(p,1)
    a = p(i,:);
    cmd = sprintf(str, a(1), a(2), a(3));
    eval(cmd);
end
```

```
perms([1 2 3])
ans:
     3     2     1
     3     1     2
     2     3     1
     2     1     3
     1     2     3
     1     3     2
```

Nd Array element access: eval

- Create parameterizable code
  - But: does not support JIT acceleration!





JIT acceleration

- Accelerates code (I guess mostly loops)
  - Not well documented, can leed to strange behavior

```
function test1
tic;
a = 2;
for i = 1:10000000
    a = a*1.00000001;
end
toc;
a = 1:2;
```

```
function test2
tic;
a = 2;
for i = 1:10000000
    a = a*1.00000001;
end
toc;
c = 1:10;
```

JIT acceleration

- Accelerates code (I guess mostly loops)
  - Avoid reassignment

```
function test1
tic;
a = 2;
for i = 1:10000000
    a = a*1.00000001;
end
toc;
a = 1:2;
```

2.31s

```
function test2
tic;
a = 2;
for i = 1:10000000
    a = a*1.00000001;
end
toc;
c = 1:10;
```

0.06s → ~40x faster!

Alternative to eval

- Useful for creating scripts
  - Uses JIT
  - Globally mounted home: ssh + screen

```
matlabexec = 'matlab -nodesktop -nosplash -r "%s; quit
force"';
cmd = sprintf(matlabexec, ml_cmd);
system(cmd);
```

Overview

- Array Memory Layout
- **Image Transformations**
  - **Gray value**
  - Coordinate
- Random Stuff

Vectorization: Matrix Mult

- Multiply all 3-vectors of a [MxNx3] with a [3x3] matrix
  - Example: colorspace transformation

```
im = single(imread('cat_sir.jpg')); %size: [631,553,3]
im = im / max(im(:));

%rgb to xyz
M = [0.5767309  0.1855540  0.1881852;
       0.2973769  0.6273491  0.0752741;
       0.0270343  0.0706872  0.9911085];

%naive: for loops
tic;
for i1 = 1:size(im,1)
      for i2 = 1:size(im,2)
            im(i1,i2,:) = M*squeeze(im(i1,i2,:));
      end
end
toc;
```

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Vectorization: Matrix Mult

- Multiply all 3-vectors of a [MxNx3] with a [3x3] matrix
  - Use shiftdim & reshape to form a 2d matrix suitable for matrix multiplication:

```
im = shiftdim(im,2);
shape = size(im);%[3,631,553]
%create a 2D matrix of col vectors
im = reshape(im, 3,[]);%[3, 631*553]
im= M*im;
%reshape to original
im = shiftdim(reshape(im, shape),1);%[631,553,3]
```

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \dots$$

Vectorization: Matrix Mult

- Multiply all 3-vectors of a [MxNx3] with a [3x3] matrix
  - equally fast: channel by channel

```
target = zeros(size(im));
target(:,:,1) = M(1,1)*im(:,:,1) + M(1,2)*im(:,:,2) + M(1,3)*im(:,:,3);
target(:,:,2) = M(2,1)*im(:,:,1) + M(2,2)*im(:,:,2) + M(2,3)*im(:,:,3);
target(:,:,3) = M(3,1)*im(:,:,1) + M(3,2)*im(:,:,2) + M(3,3)*im(:,:,3);
```

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Vectorization: Matrix Mult

- Timings
  - For loops: **20.77**
  - Shiftdim & Reshape: **0.007**
  - Channel wise: **0.01**

Overview

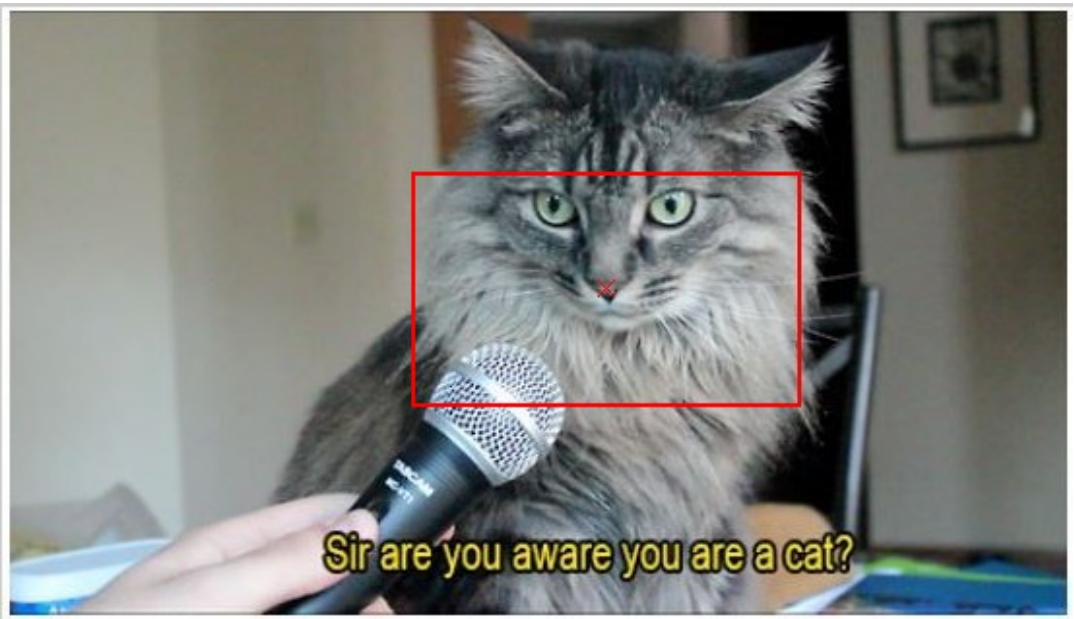
UNI FREIBURG

- Array Memory Layout
- **Image Transformations**
  - Gray value
  - **Coordinate**
- Random Stuff

Patch-extract: interpn, ndgrid

- (linear) Transformations to extract an image patch

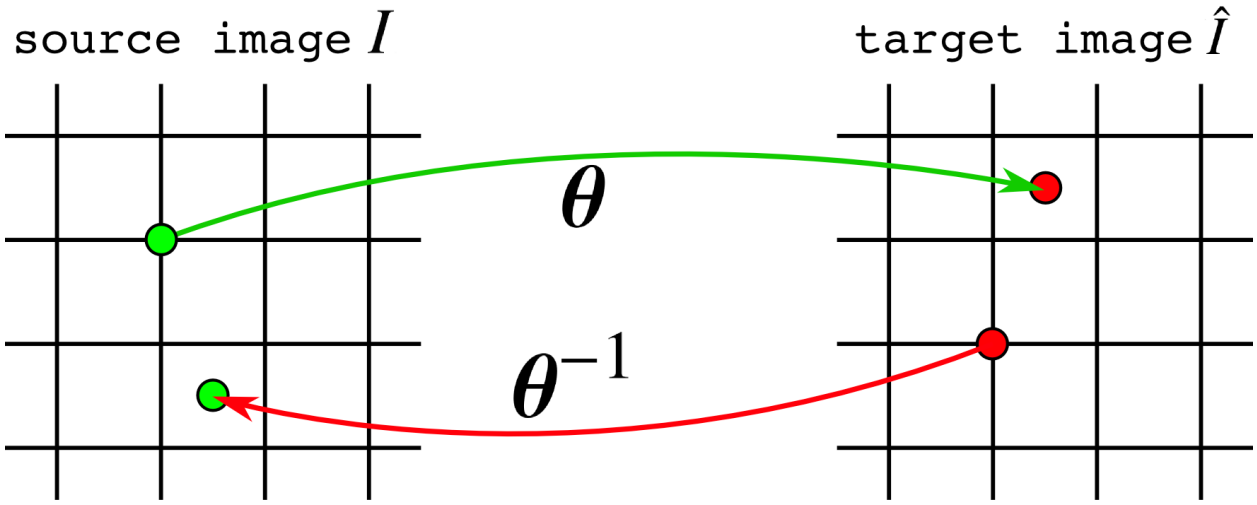
source

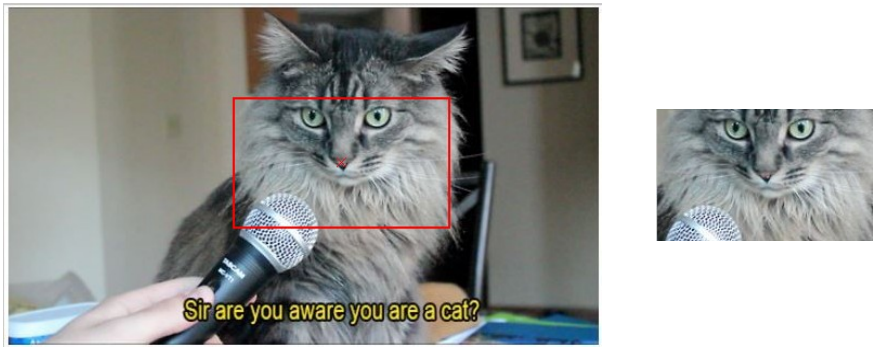


target

Patch-extract: interpn, ndgrid

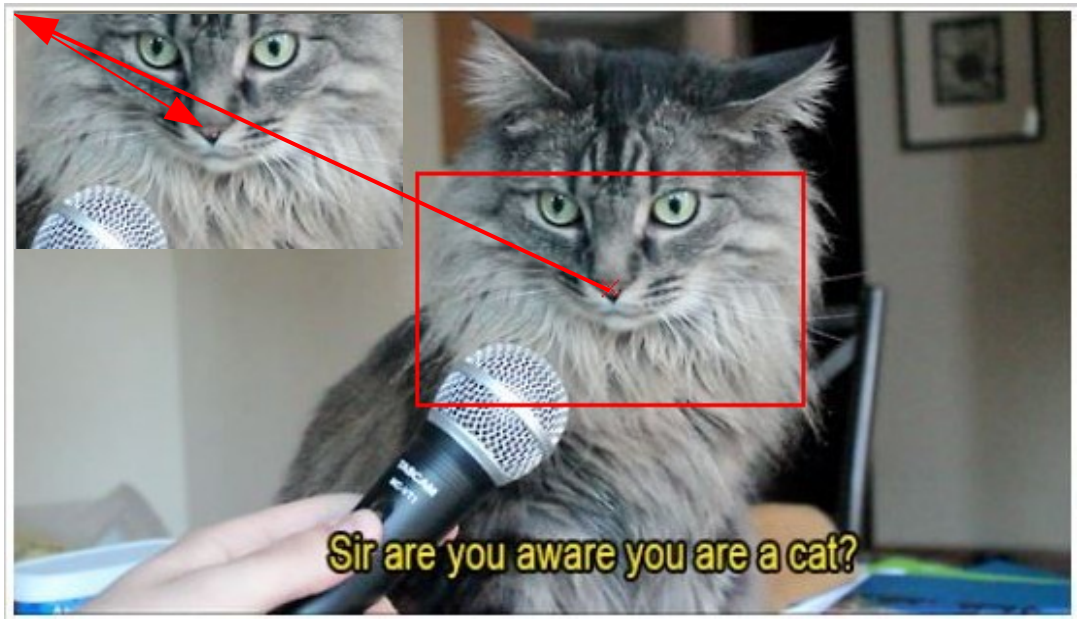
- (linear) Transformations to extract an image patch





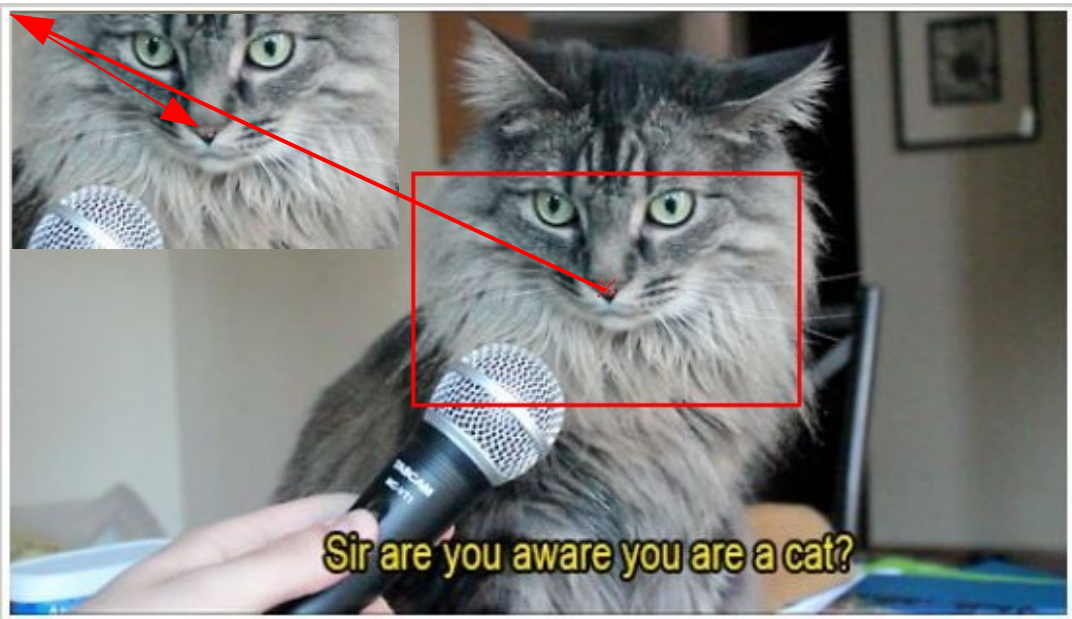
Patch-extract: interpn, ndgrid

- (linear) Transformations to extract an image patch



Patch-extract: interpn, ndgrid

- (linear) Transformations to extract an image patch




```
%to origin
M1 = [  1 0 -y;
        0 1 -x;
        0 0 1];

%origin to center of target
M2 = [  1 0 target_size(1)/2;
        0 1 target_size(2)/2;
        0 0 1];
```

Patch-extract: interpn, ndgrid

- Ndgrid: create grid indices for arbitrary dimensions

```
[X,Y] = ndgrid(1:3,1:4)

X:
     1     1     1     1
     2     2     2     2
     3     3     3     3

Y:
     1     2     3     4
     1     2     3     4
     1     2     3     4
```

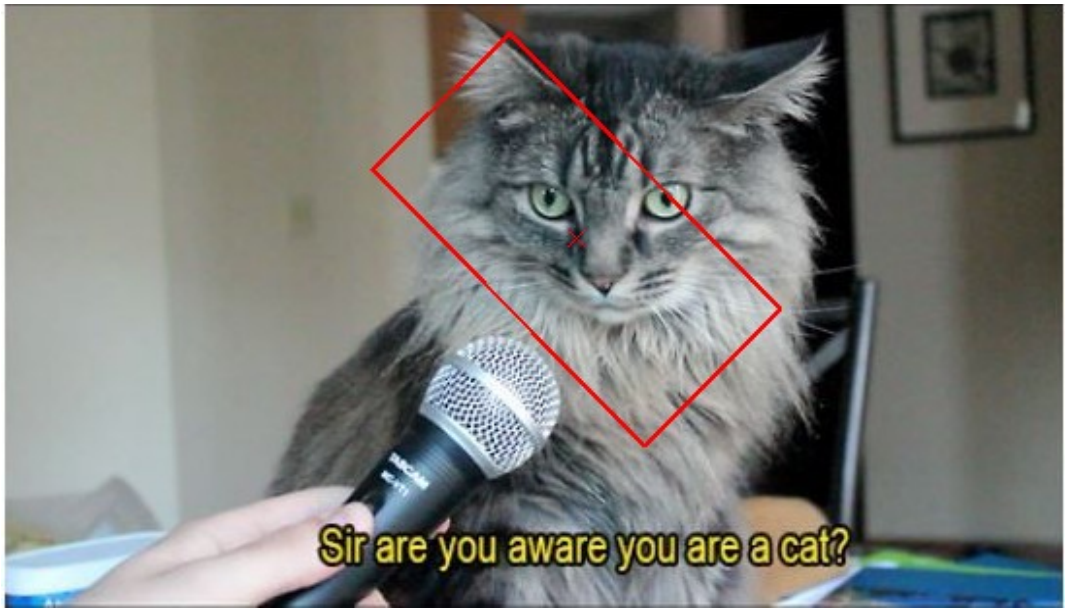
Patch-extract: interpn, ndgrid

- Putting stuff together

```
%get inverse transformation
M = pinv(M2*M1);

%create grid
[X,Y] = ndgrid(1:target_size(1), 1:target_size(2));

%apply transformation
XT = M(1,1)*X + M(1,2)*Y + M(1,3);
YT = M(2,1)*X + M(2,2)*Y + M(2,3);

%read out source with interpolation
target = zeros([target_size, size(im,3)]);
for ch = 1:size(im,3)
      target(:,:,ch) = interpn(im(:,:,ch), XT, YT, 'bicubic',0);
end
```

Patch-extract: interpn, ndgrid

- Adding Rotations




```
R = [   cos(alpha), -sin(alpha), 0;
        sin(alpha), cos(alpha), 0;
        0, 0, 1];

M = pinv(M2*R*M1);
```

Patch-extract: interpn, ndgrid

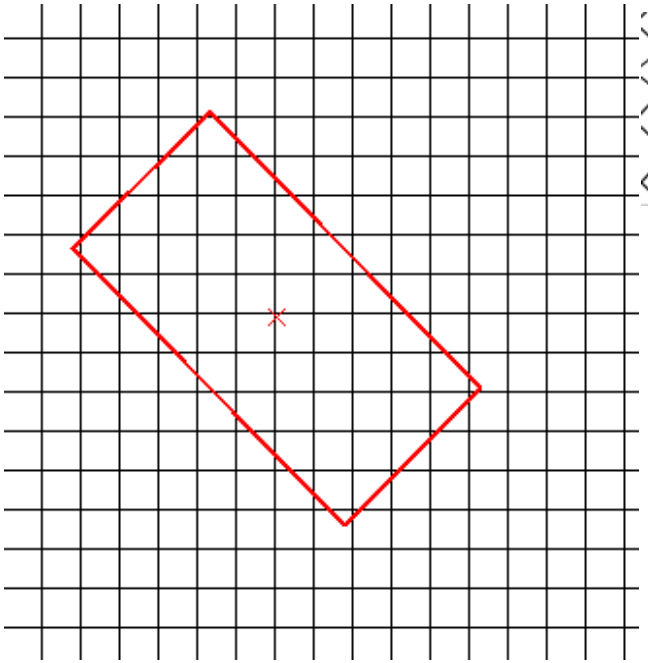
- Border treatment: mirroring

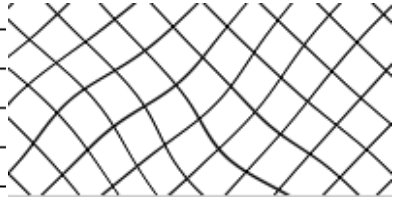
Artifacts due to Screencapture / libre office

```
%mirror?
XT(XT < 1) = 2 - XT(XT < 1);
YT(YT < 1) = 2 - YT(YT < 1);
XT(XT > size(im,1)) = 2*size(im,1) - XT( XT > size(im,1));
YT(YT > size(im,2)) = 2*size(im,2) - YT( YT > size(im,2));
```

Patch-extract: interpn, ndgrid

- Adding wobble





```
%wobble
u = randn(3,3);
v = randn(3,3);
U = imresize(u,size(X),'bicubic');
V = imresize(v,size(X),'bicubic');

%  put all together
XT = M(1,1)*X + M(1,2)*Y + M(1,3) + wobble * U;
YT = M(2,1)*X + M(2,2)*Y + M(2,3) + wobble * V;
```

Overview

- Array Memory Layout
- Image Transformations
- **Random Stuff**

Startup.m

- Automatically load config file on startup
- .bashrc:
  - **`MATLABPATH=$MATLABPATH:/path/to/startup.m/`**
- Startup.m:

```
%add all paths recursively that are rooted here
if(~isdeployed)
	addpath(genpath('/home/maid/phd/trunk/matlab/'));
else
	disp('In deployed mode.');
end

%load standard- config file: generative model detection
global config;
s = '/home/maid/phd/trunk/matlab/config.mat';
load(s);
fprintf('loaded global config from %s\n',s);
```

Global: config

```
config =

CROSSVALIDATION: 1
CVALS: [1x12 double]
servers: {'dacky'  'frieda'  'william'}
numJobs: [15 15 10]
verbose: 0
fileLocation:
'/home/maid/phd/trunk/matlab/config.mat'
debug: 0
trainingBaseFolder:'/pathto/somefolder/'
recomputeFeatures: 0
computeFeatures: @hogCellsForVolume_mex
vibezVersion: @myVibez3
```

Distribute values to struct

```
%declare anonymous functions
cellexpand = @(x) x{:};
numexpand = @(x) cellexpand(num2cell(x));

clear udets;
%init struct, all accepted = 0
[udets(1:1000).accepted] = deal(0);
%distribute array to struct
[udets.id] = numexpand(1:1000);
```

```
udets(1):
    accepted: 0
          id: 1
udets(2):
    accepted: 0
          id: 2
```

Standard function arguments

```
function res = tut_varargin(a1,a2, varargin)

    %std. values
    optargs = {eps, 10, @zeros};

    %replace std. values
    optargs(1:length(varargin)) = varargin;

    %give nicer variable names
    [tol, its, init_func] = optargs{:};
```

combvec & bsxfun

- Create corners of a cuboidal box

```
shape = [120, 80, 90]';
corners = bsxfun(@times, shape,...
    combvec([1,0], [1,0], [1,0]))

corners:
   120     0   120     0   120     0   120     0
    80    80     0     0    80    80     0     0
    90    90    90    90     0     0     0     0

combvec([1,0], [1,0], [1,0])
ans:

     1     0     1     0     1     0     1     0
     1     1     0     0     1     1     0     0
     1     1     1     1     0     0     0     0
```

cell argument lists

- Cell expansion using {:}
  - Generate arbitrary long list of arguments
  - Does not need to be of uniform type

```
%cell-argument list
c = repmat({[0 1]},1,3); %c: 1x3 cell
res = combvec(c{:});
```

ismember

- Pick out a subset

```
%pick out a subset: ismember
[dets(1:100).ch] = numexpand(randi(10,1,100));
subset = dets(ismember([dets.ch], [1,4,5]));
```

```
[subset.ch]
ans:
  Columns 1 through 21
     5     5     1     5     4     4     4     5     1     4     1     4     5     5     1     5     1     5     1     4     5
  Columns 22 through 29
     4     5     1     1     5     1     4     4
```

histc

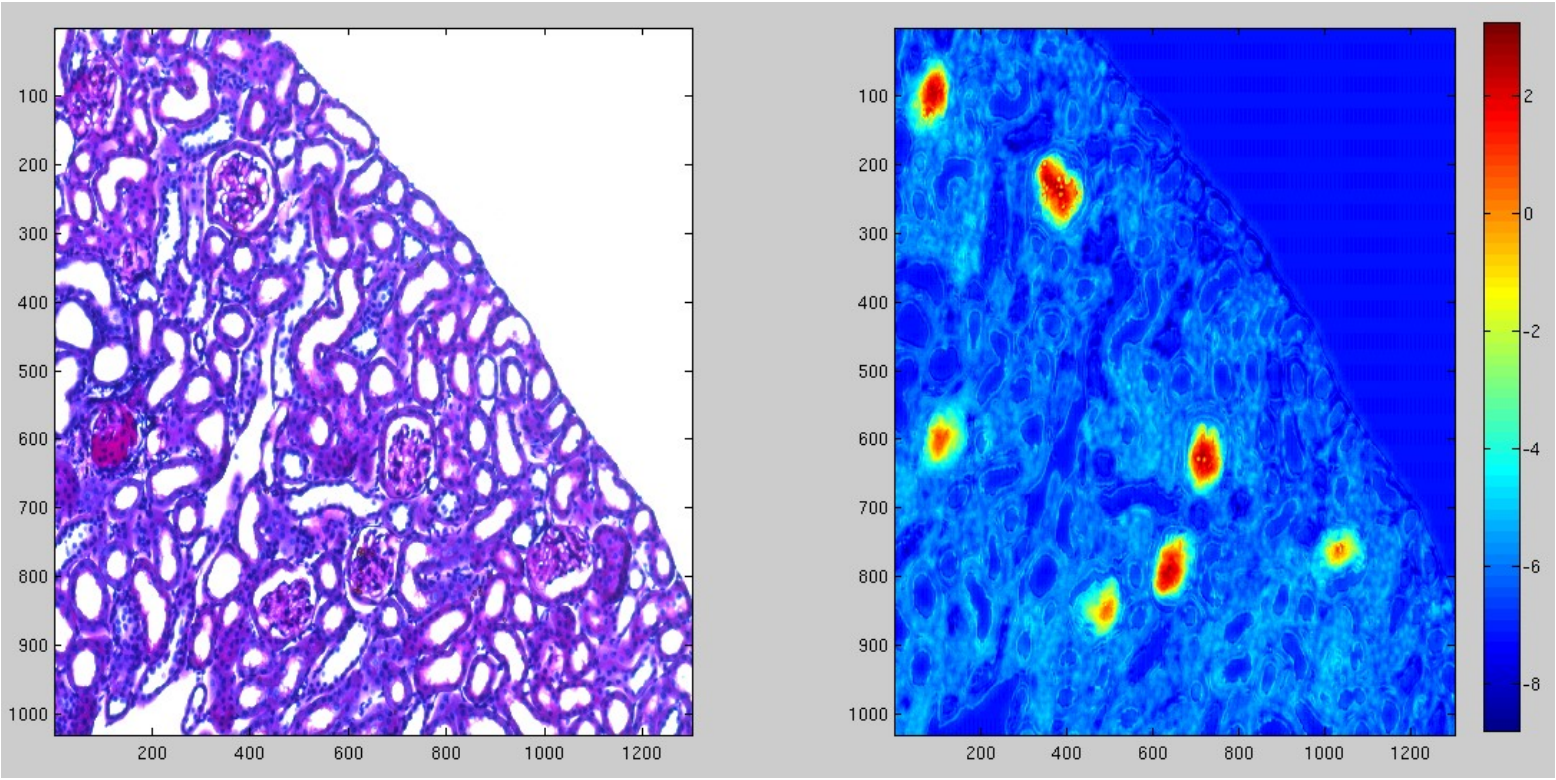
- Count occurences

```
%count occurences
count = histc([dets.ch], unique([dets.ch]));

unique([dets.ch]):

      1    2    3    4    5    6    7    8    9   10

count:

      9   12   16    9   11    7   10    5   13    8
```

Local maxima

- With morphological operations

```
se = ones([mindist_x, mindist_y]);
res = imdilate(data, se);
lmax = find(data == res);
```



Find Dependencies of .m files

UNI FREIBURG

```
[flist, plist] =
matlab.codetools
.requiredFilesAndProducts('getInvField2.m')
```

flist':

```
'/home/maid/phd/trunk/matlab/3rdparty/ann_mwrapper/annquery.m'
'/home/maid/phd/trunk/matlab/3rdparty/ann_mwrapper/private/ann_
mex.mexa64'
 /home/maid/phd/trunk/matlab/3rdparty/ann_mwrapper/private/seto
pts.m'
 /home/maid/phd/trunk/matlab/3rdparty/ann_mwrapper/private/sho
wdoc.m'
 /home/maid/phd/trunk/matlab/helpers/getInvField2.m'
```

plist(1):

```
        Name:  MATLAB'
     Version:  8.3'
ProductNumber: 1
```

plist(2):

```
        Name:  Image Processing Toolbox'
     Version:  9.0'
ProductNumber: 17
```

JAVA integration

- Write Excel tables
  - http://www.mathworks.com/**matlabcentral**/fileexchange/38591-xlwrite--generate-xls-x--files-without-excel-on-mac-linux-win

```
sheet1 = {'col1', 'col2', 'col3'; 4 5 6; 'asdf' 4 'fgf'};
sheet2 = {'col1', 'col2', 'col3'; 4 5 6; 'asdf' 4 'fgf'};

xlwrite('tut_demo.xls', sheet1, 'page1');
xlwrite('tut_demo.xls', sheet2, 'Loss');
```

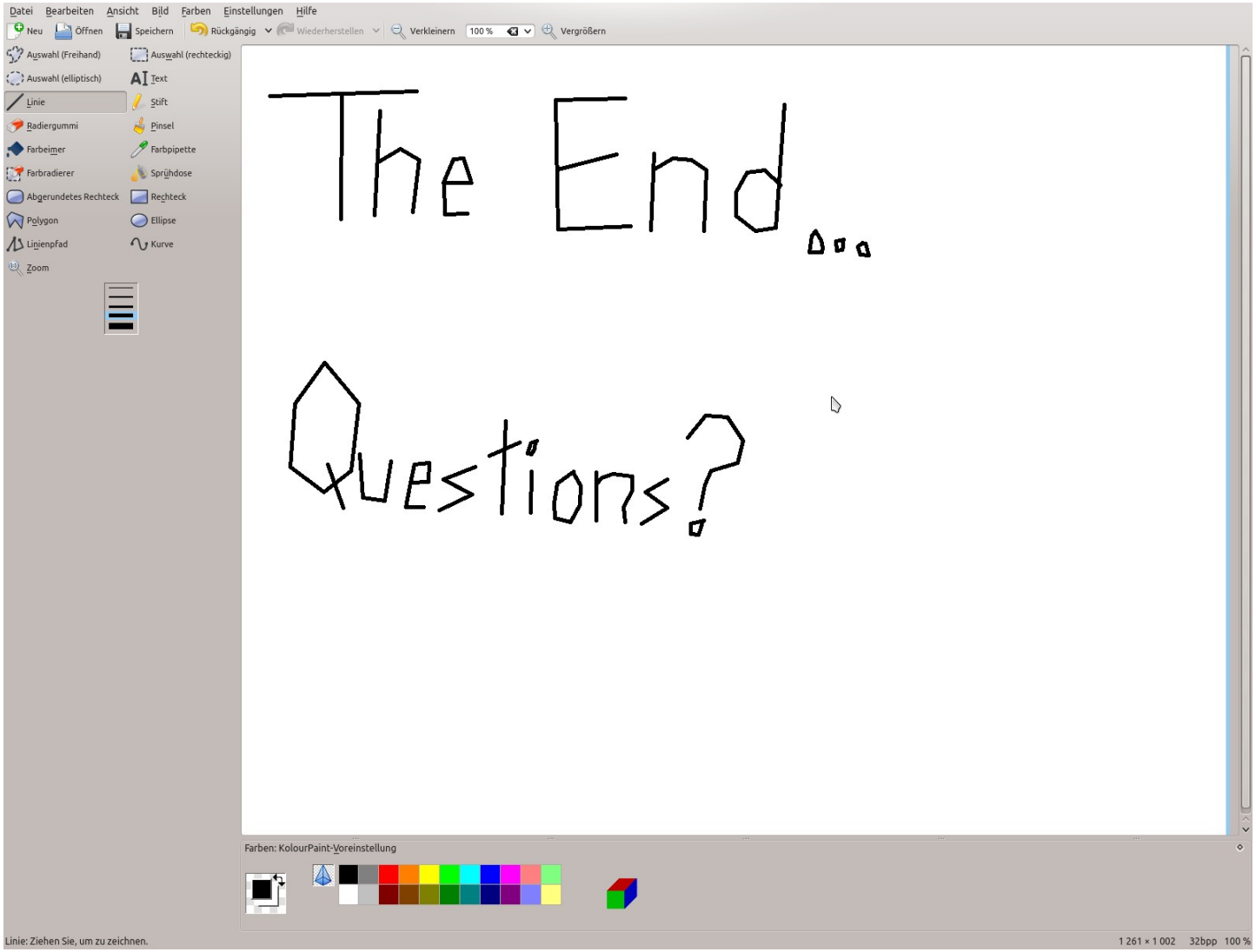
JAVA integration

- Transparent variable conversion
- Java.awt.robot

```
robot = java.awt.Robot;
robot.mouseMove(x,y);
robot.keyPress(asciiNum);
```

http://de.mathworks.com/matlabcentral/fileexchange/32971-dynamic-search-box/content/java_robot.m

Fin.


Datei Bearbeiten Ansicht Bild Farben Einstellungen Hilfe
Neu Öffnen Speichern Rückgängig Wiederherstellen Verkleinern 100 % Vergrößern
Auswahl (Freihand)
Auswahl (rechteckig)
Auswahl (elliptisch)
Text
Linie
Stift
Radiergummi
Pinsel
Farbeimer
Farbpipette
Farbradierer
Sprühdose
Abgerundetes Rechteck
Rechteck
Polygon
Ellipse
Linienpfad
Kurve
Zoom
The End...
Questions?
Farben: KolourPaint-Voreinstellung
Linie: Ziehen Sie, um zu zeichnen.
1 261 × 1 002 32bpp 100 %