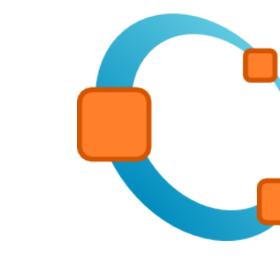
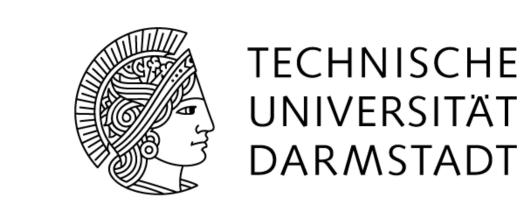




Examples for teaching mathematical programming using Octave

Alf Gerisch (TU Darmstadt) and Markus Köbis (Uni Halle) and Helmut Podhaisky (Uni Halle)





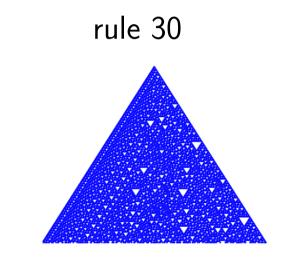
Abstract

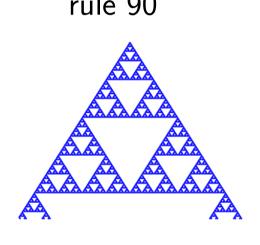
We believe that mathematical programming can be taught using surprising and nontrivial examples. Short programs shall visualise an idea. To learn how to write correct code is easier with discrete problems where rounding errors cannot conceal bugs. The lack of certain data structures (lists with $\mathcal{O}(1)$ append, queues, ...) in Octave leads to uglier code or wrong asymptotic complexity for some problems (shortest paths, minimal spanning trees, ...). Visit https://github.com/hpodhaisky/OctConf for download.

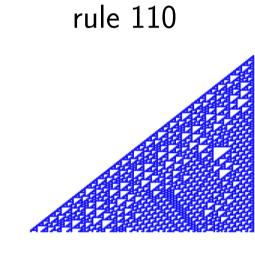
Cellular Automatons

```
rule = [30, 90, 110]; n = 150;
for r = rule
 M=zeros(n,2*n); M(1,n) = 1;
 for i=2:n
   for j=2:2*n-1
     M(i,j)=bitget(r, 1+M(i-1,j-1:j+1)*[4 2 1]');
 spy(M,5); axis off; axis tight;
```

The 256 different functions $f_r: \{0,1\}^3 \to \{0,1\}, f_r: (x_{i-1}^i, x_i^i, x_{i+1}^i) \mapsto x_i^{i+1}$ are encoded with one value $r \in \{0, \dots, 255\}$. The evolution of the one-dimensional system over time is from top to bottom. The graphics with spy is slow.





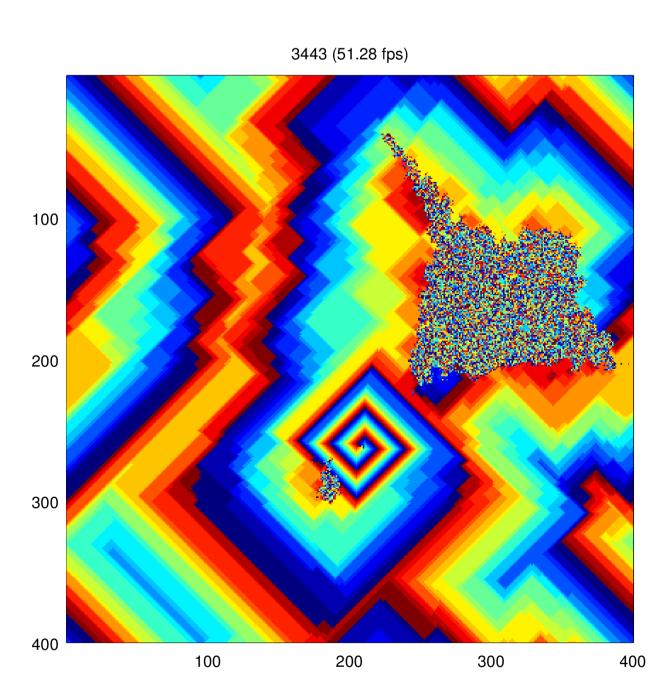


Generated with automaton.m. There is chaos, structure and even universal computation [1].

cells.m

```
k=20; n=400;
self = reshape(1:n*n,n,n);
left = self(:,[n,1:n-1]);
right = self(:,[2:n,1]);
     = self([n,1:n-1],:);
down = self([2:n,1],:);
     = floor(k*rand(n,n));
     = imagesc(Z); axis square; tic;
for gen = 1:10000
 G = mod(Z(self)+1,k);
  i = (G==Z(down))|(G==Z(up))|(G==Z(left))|(G==Z(right));
 Z(i)=G(i); set(h, 'cdata', Z); e = toc;
 title(sprintf('%d (%5.4g fps)',gen, gen/e)); drawnow
```

A cell in state z is eaten by a neighbouring cell in state z + 1. A level of indirection makes the computation concise as well as fast.



in which rotating spirals arise out of chaos.

Reaction diffusion equations

function grayscott

```
m = 150; L = 2; tau = 0.1; u = ones(m,m); v = zeros(m,m);
 [xx, yy] = meshgrid(linspace(0,L,m));
 u(m/2+(1:20), m/2+(1:20)) = 1/2+0.1*(rand(20,20)-1);
v(m/2+(1:20), m/2+(1:20)) = 1/4+0.05*(rand(20,20)-1);
 for k=0:1000000
   [du,dv] = f(u, v); u = u+tau*du; v = v+tau*dv;
   if mod(k,50)==0, contourf(xx,yy,u,linspace(0.1,0.9,4))
    title(['time t=',num2str(tau*k)]);
    axis equal; axis square; axis tight; axis off; drawnow
 function [du,dv]=f(u,v)
m = 150; ip = [2:m,1]; im = [m,1:m-1]; Du = 2e-5; Dv = 1e-5;
 L = 2; h = L/m; F = 0.03; k = 0.055; r = u.*v.^2;
```

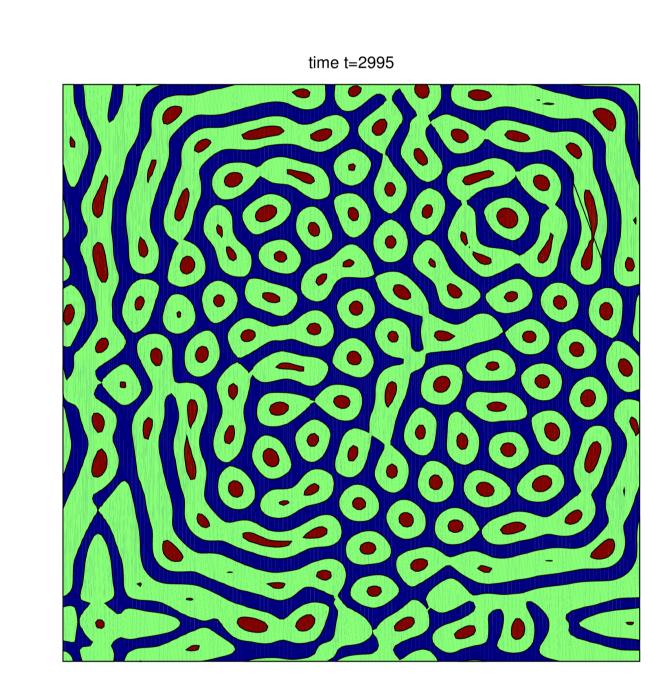
grayscott.m

Solving $u_t = D_u \Delta u - uv^2 + F(1-u)$, $v_t = D_v \Delta v + uv^2 - (F+k)v$ with periodic boundary conditions using central differences for the Laplacians and the $i \in x$ explicit Euler method for integration.

diffu = $Du/h^2*(u(ip,:)+u(im,:)+u(:,ip)+u(:,im)-4*u);$

diffv = $Dv/h^2*(v(ip,:)+v(im,:)+v(:,ip)+v(:,im)-4*v);$

du = diffu - r + F*(1-u); dv = diffv + r - (F+k)*v;



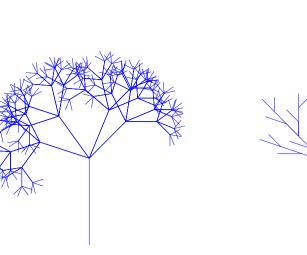
Self-organization in the Gray-Scott reaction-diffusion system [3], generated with grayscott.m.

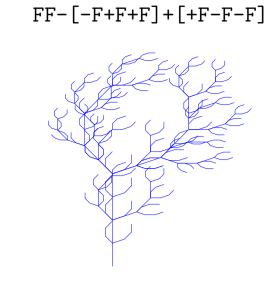
Lindenmayer's L-systems

```
function lsystem(rule, scale, phi, psi, depth)
G = F(rule, scale, phi, psi, depth, 0, 0, 1j, []);
plot(real(G), imag(G))
function [G,x,dx,k] = F(r, s, phi, psi, gen, k, x, dx, G)
if gen==0, seg = [x, x+dx]; G = [G, seg]; x = seg(2);
 while k < length(r)
   k = k + 1;
    switch r(k)
    case 'F'; [G,x,dx,^{\sim}] = F(r,s,phi,psi,gen-1,0,x,dx,G);
    case '+'; dx = exp(phi*1j) * dx;
    case '-'; dx = exp(-psi*1j) * dx;
    case '['; [G, \tilde{,}, \tilde{,}k] = F(r,s,phi,psi,gen,k,x,s*dx,G);
    case ']'; G = [G, nan]; break;
    end
  end
```

Adapted from [4]. We use complex multiplication for rotation and recursion to Life in a cyclic world [2], generated with cells.m. This is a snapshot of a movie avoid maintaining a stack of coordinates. Line segments are separated by NaNs for efficient plotting.

F[+F][-F][++F][--F]



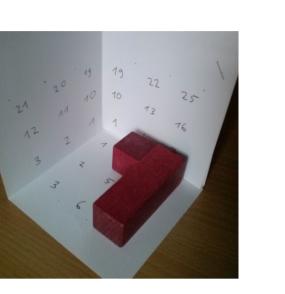


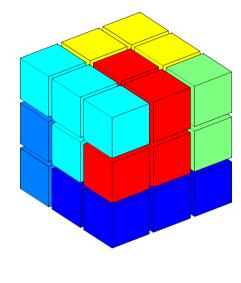
Plants as generated with lsystem.m, cf. [4].

Soma cube

function soma







Soma cube. Filling the space with properly aligned pieces is an instance of a set-cover problem: Given a set of subsets, $s_i \subset \Omega$, find selection x such that $\dot{\cup} \ s_i = \Omega$. There are 480 distinct ways to assemble the cube [5] and soma.m computes them in a few seconds.

```
pieces = {[1 2 3 4],[1 2 4],[1 2 3 5], [1 2 5 6],...
  [1 \ 2 \ 4 \ 10], [1,2,4,11], [1,2,4,13]; % 3x3x3 = 27
for k=1:length(pieces)
  if k==1, T1 = pieces\{k\};
  else, T1 = rotations(pieces{k}); end
  for l=1:size(T1,1)
    T2=shifts(T1(1,:));
    for i=1:size(T2,1)
      c = zeros(34,1); c([T2(i,:),27+k]) = 1;
      A(:,col)=c; col=col+1;
tic, X=backtrack(A,[],1:size(A,2)), toc
function t = normal(xyz)
x=xyz(1,:);y=xyz(2,:);z=xyz(3,:);
t=sort(sub2ind([3,3,3],x-min(x)+1,y-min(y)+1,z-min(z)+1));
function T = rotations(t)
T=[t]; l=1;
Dx = [1 \ 0 \ 0; \ 0 \ 0 \ -1; \ 0 \ 1 \ 0];
Dy=[0 \ 0 \ -1; \ 0 \ 1 \ 0; \ 1 \ 0];
Dz=[0 -1 0; 1 0 0; 0 0 1];
G1={eye(3),Dz,Dz^2,Dz^3,Dy,Dy^3};
G2=\{eye(3),Dx,Dx^2,Dx^3\};
[x,y,z] = ind2sub([3,3,3],t);
for g1=G1
  for g2=G2
    D=g1\{:\}*g2\{:\}; s=normal(D*[x;y;z]);
    if all(any(T^=repmat(s,l,1),2)), T=[T;s]; l=l+1; end
function T = shifts(t)
T=[]; [x,y,z]=ind2sub([3,3,3],t);
for i=max(x):3
  for j=max(y):3
    for k=max(z):3
      s=sub2ind([3,3,3],x+i-max(x),y+j-max(y),z+k-max(z));
      T=[T;s];
```

function X = backtrack(A,x,active)

if all(b==0), X=x; somadraw(A,x);

bs = find(b); k = bs(criticalb);

[egal, criticalb] = min(sum(A(b,active),2));

n = length(active); X = [];

 $b=^{(sum(A(:,x),2))};$

```
for w = active(find(A(k,active)==1))
  an=active(all((A(:,active) & repmat(A(:,w),1,n))==0));
 X=[X,backtrack(A,[x;w],an)];
```

We calculate valid placements for each piece in an empty cube (using rotations [6] and shift, whereas the L shaped piece No. 1 is not rotated to fix the rotational symmetry). The physical space $3 \times 3 \times 3$ is extended by seven components to mark the number of the piece, leading to Ax = b with $A \in \{0,1\}^{34 imes 550}$, $x \in \{0,1\}^{550}$ and $b = [1,\ldots,1]^ op$ which is solved by backtracking.

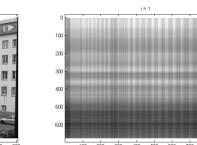
```
somadraw.m
function somadraw(A,u)
Vertices = [ 0 0 0; 0 0 1; 0 1 0; 0 1 1
            1 0 0; 1 0 1; 1 1 0; 1 1 1 ];
Faces = [1 2 6 5; 1 2 4 3; 1 3 7 5;
        2 4 8 6; 3 4 8 7; 5 6 8 7 ];
cm = jet(7); view(3); axis([0 3 0 3 0 3]);
axis equal; axis off; cla
for k=u'
  f=(1:7)*A(28:end,k);
  for i = find(A(1:27,k))';
   [x,y,z] = ind2sub([3,3,3],i);
    patch('Vertices',0.9*Vertices+repmat([x y z]-1,8,1), ...
     'Faces', Faces, 'EdgeColor', 'k',...
     'FaceVertexCData',cm(f,:),'FaceColor','flat');
  end, drawnow
```

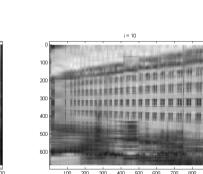
We're drawing complete and incomplete solutions during the calculation.

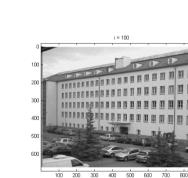
Singular value decomposition

Data compression









Decomposition of a given bitmap and its reconstructions using 1, 10 and 100 modes using datacompression.m

```
datacompression.m
function datacompression(file)
A = double(imread(file));
[m,n] = size(A);
[U,S,V] = svd(A);
St = zeros(size(S));
for i = 1 : min(m,n)
    St(i,i) = S(i,i);
    At = U*St*V';
                           % reconstruction using
    imagesc(At);axis equal; % just a few singular values
   title(sprintf('i = %d',i));
```

We handle the grayscale image as matrix input and reconstruct it step-by-step using its principal components aquired by Octave's svd.m routine.

Face recognition



Adjusting the face orientation and eye position using graphical user interface

adjustportrait.m

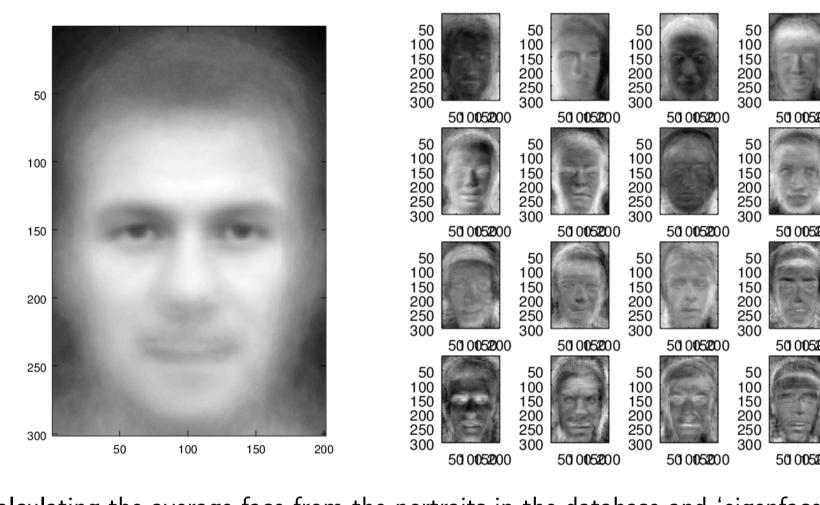
```
function adjustportraits(flag)
if nargin==0, flag = 'start'; end
switch flag
 case 'start' % Initialize GUI
  f = figure('Units','Normalized','DefaultUicontrolUnits',..
          'Normalized', 'Position', [.1 .1 .8 .8]);
  ud.axes(1) = axes('Parent',f,'Position',[.05 .05 .4 .9]);
  ud.axes(2) = axes('Parent',f,'Position',[.8 .05 .18 .5]);
  ud.button(1) = uicontrol(f,'Position',[.55 .9 .2 .05],...
```

```
'String', 'right eye', 'FontSize', 20, 'Callback', ...
      'adjustportraits(''sr'')');
ud.check(1) = uicontrol(f,'Position',[.8 .9 .05 .05],...
  'Style', 'Text', 'FontSize', 20, 'String', '');
ud.button(2) = uicontrol(f, 'Position', [.55 .8 .2 .05], ...
  'String', 'left eye', 'FontSize', 20, 'Callback', ...
      'adjustportraits(''sl'')');
```

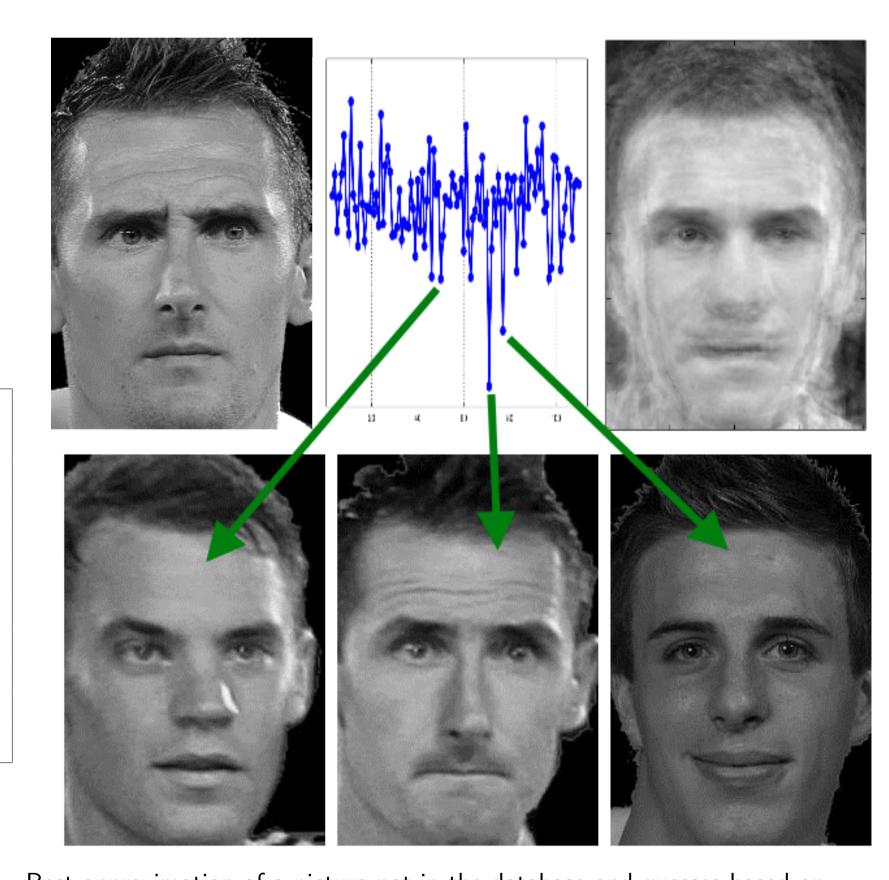
ud.check(2) = uicontrol(f,'Position',[.8 .8 .05 .05],... 'Style','Text','FontSize',20,'String',''); ud.button(4) = uicontrol(f, 'Position', [.55 .6 .2 .05], ... 'String','Okay','FontSize',20,'Callback',... 'adjustportraits(''calculate'')');

Building the graphical user interface with uicontrol.m, code snippet

ud.button(5) = uicontrol(f,'Position',[.55 .5 .2 .05],...



Calculating the average face from the portraits in the database and 'eigenfaces'



Best approximation of a picture not in the database and guesses based on eigenface estimation

References

https://en.wikipedia.org/wiki/Rule_110

https:

//en.wikipedia.org/wiki/Cyclic_cellular_automaton

- John Pearson: Complex pattern in a simple system, Science 1993, 189–192
- Higham, Higham: Matlab Guide (2nd ed.), SIAM 2005
- https://en.wikipedia.org/wiki/Soma_cube
- https://en.wikipedia.org/wiki/Octahedral_symmetry
- https://de.wikipedia.org/wiki/Liste_der_ deutschen_Fu%C3%9Fballnationalspieler/*
- Muller, Magaia, Herbst: Singular Value Decompostion, Eigenfaces, and 3D Reconstructions, SIAM REVIEW (46) 518-545, 2004