Writing faster and better Matlab code

Preallocate arrays

- Create storage for array results before filling it.

```
% Matlab script -- compute sin(x) for values from 0:100000

clear all

% No preallocation
x = linspace(0, 100, 100000);
tic;
for i = 1:length(x)
  y(i) = sin(x(i));
end
t = toc;
fprintf('No preallocation.  Computed %d values in %f seconds\n', length(x), t)

clear all

% With preallocation
x = linspace(0, 100, 100000);
y = zeros(size(x));
tic;
for i = 1:length(x)
  y(i) = sin(x(i));
end
t = toc;
fprintf('With preallocation.  Computed %d values in %f seconds\n', length(x), t)
```

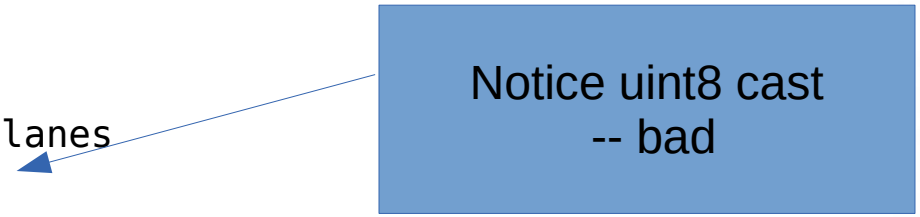
Preallocate arrays

```
>> PreallocateArrays
No preallocation.  Computed 100000 values in 0.045132 seconds
With preallocation.  Computed 100000 values in 0.008846 seconds
```

- Preallocation is about 5 -- 7x faster.
- With no preallocation, Matlab must continually spend time finding new storage for the growing output array.
- With preallocation, no time is wasted resizing the output array at each loop iteration.

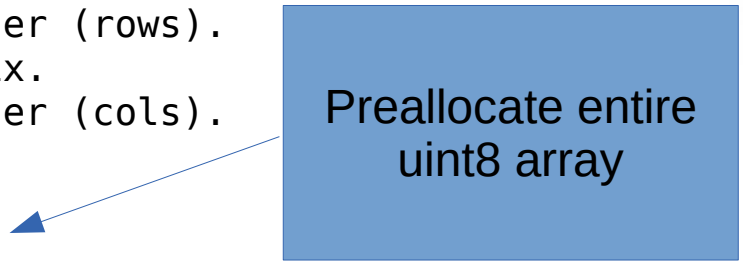
Preallocate arrays of specific type

```
% Matlab script -- Draw white circle on black blackground

clear all

% No preallocation
Rows = 250;   % Number of rows in image matrix.
Rc = Rows/2;  % Where to place the image center (rows).
Cols = 250;   % Number of cols in image matrix.
Cc = Cols/2;  % Where to place the image center (cols).
Planes = 3;   % Three color planes
Rad = 50;     % Radius of circle to draw
tic;
for r = 1:Rows
  for c = 1:Cols
    for p = 1:Planes
      if norm([r - Rc, c - Cc]) < Rad
        % white circle is 255 on all planes
        myimage(r, c, p) = uint8(255);
      else
        myimage(r, c, p) = uint8(0);
      end
    end
  end
end
t = toc;
figure
image(myimage)
fprintf('No preallocation.  Created matrix in %f seconds\n', t)
```



Preallocate arrays of specific type

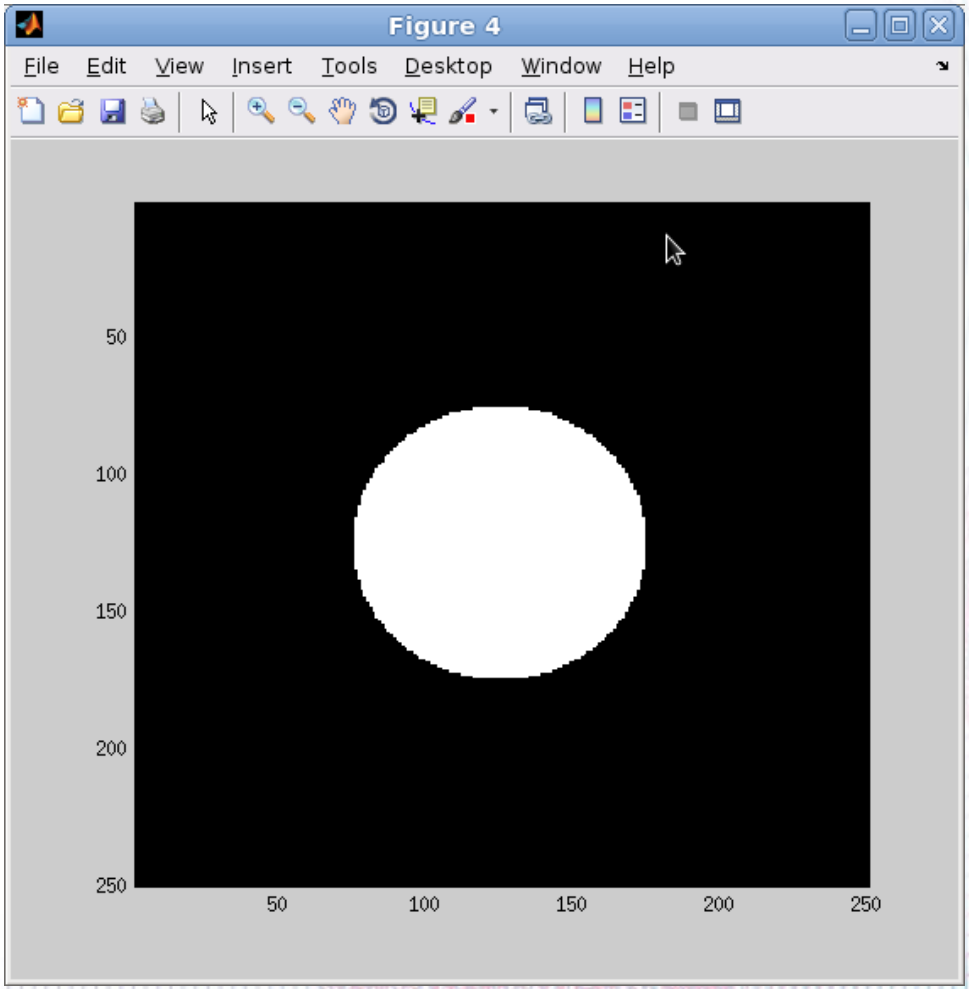
```
% Draw white circle on black blackground
clear all

% With preallocation
Rows = 250;   % Number of rows in image matrix.
Rc = Rows/2;  % Where to place the image center (rows).
Cols = 250;   % Number of cols in image matrix.
Cc = Cols/2;  % Where to place the image center (cols).
Planes = 3;   % Three color planes
Rad = 50;     % Radius of circle to draw
myimage = zeros(Rows, Cols, Planes, 'uint8');
tic;
for r = 1:Rows
  for c = 1:Cols
    for p = 1:Planes
      if norm([r - Rc, c - Cc]) < Rad
        % Draw white circle
        myimage(r, c, p) = 255;
      end
    end
  end
end
t = toc;
figure
image(myimage)
fprintf('With preallocation.  Created matrix in %f seconds\n', t)
```



Comparison

```
>> PreallocateArraysType
No preallocation.  Created matrix in 0.395536 seconds
With preallocation.  Created matrix in 0.187897 seconds
```

- In this case, 2x speedup with preallocation.
- Matlab gives you the ability to create arrays of specific types using zeros(), ones(), eye(), etc.



Vectorize your code

- Vectorize = Avoid “for” loops when possible. Use built-in primitives which accept array inputs.

```
% Matlab script -- compute sin(x) for values from 0:100000

clear all

% No vectorization (but with preallocation)
x = linspace(0, 100, 100000);
y = zeros(size(x));
tic;
for i = 1:length(x)
  y(i) = sin(x(i));
end
t = toc;
fprintf('No vectorization.  Computed %d values in %f seconds\n', length(x), t)

clear all

% Vectorized
x = linspace(0, 100, 100000);
tic
y = sin(x);
t = toc;
fprintf('Vectorized.  Computed %d values in %f seconds\n', length(x), t)
```

Effects of vectorization

- Factor of 4 speedup with vectorized code.

```
>> Vectorize
No vectorization.  Computed 100000 values in 0.009425 seconds
Vectorized.  Computed 100000 values in 0.002328 seconds
```

- Effect is much more pronounced in Octave – 200x speedup

```
octave:3> Vectorize
No vectorization.  Computed 100000 values in 1.140878 seconds
Vectorized.  Computed 100000 values in 0.005408 seconds
```

- This is because Matlab has a very smart JIT – just in time compiler – which can recognize and vectorize simple for loops.

Use “meshgrid” to create {x, y} grids

```
>> [X, Y] = meshgrid(-2:2, -3:3)

X =

    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2

Y =

    -3    -3    -3    -3    -3
    -2    -2    -2    -2    -2
    -1    -1    -1    -1    -1
     0     0     0     0     0
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
```

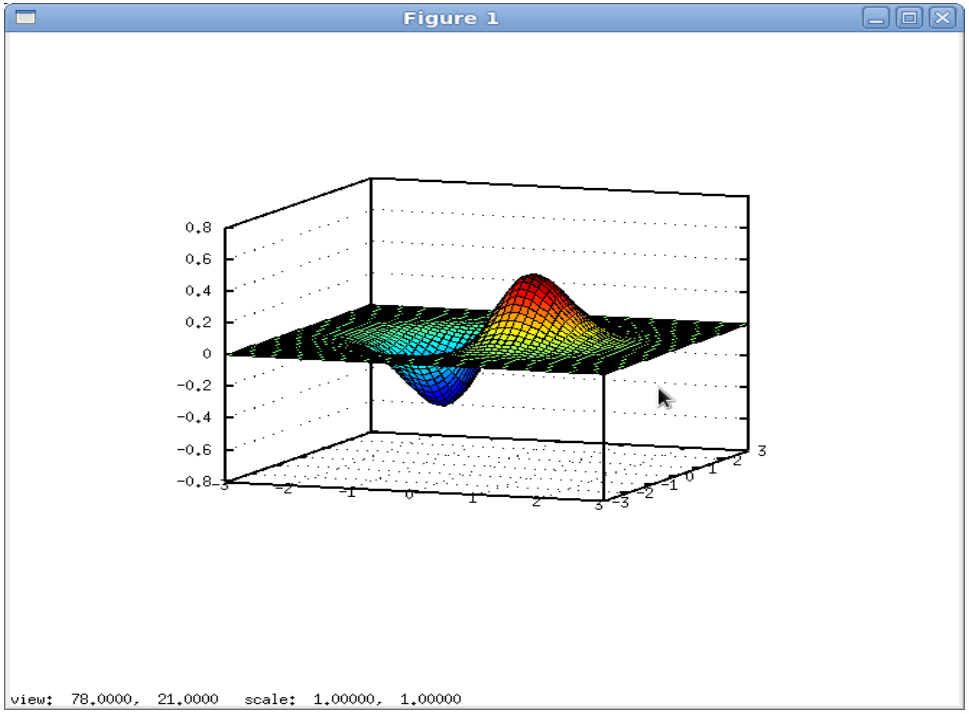
- Meshgrid takes vectors, returns matrices.
- X, Y pairs form coordinate set [x(i, j), y(i, j)]
- Example: [x(1,1),y(1,1)] = [-2,-3]
- Example: [x(5,5),y(5,5)] = [2, 1]

Example of meshgrid use

- Create plot of function

$$z(x, y) = x e^{-(x^2 + y^2)}$$

- Non-vectorized algorithm would compute z(i, j) using two for loops.
- Note: Time vs. memory trade-off.




```
u = linspace(-3, 3, 50);
v = linspace(-3, 3, 50);
[x, y] = meshgrid(u, v);
z = x .* exp(-x.^2 - y.^2);
surf(x, y, z)
```

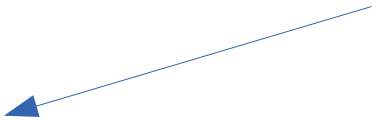
Use logical indexing

- Logical indexing instead of for loops is a different aspect of vectorization.
- Here, we find all negative elements of A.
- Return is a mask.

```
>> A = randn(3,4)

A =

    0.1832    0.3071    0.2614   -0.1461
   -1.0298    0.1352   -0.9415   -0.5320
    0.9492    0.5152   -0.1623    1.6821

>> lt = (A < 0)

lt =

     0     0     0     1
     1     0     1     1
     0     0     1     0

>> A(lt)

ans =

   -1.0298
   -0.9415
   -0.1623
   -0.1461
   -0.5320
```

Vectorized circle drawing program

```
% Matlab script -- Draw white circle on black blackground

clear all

% With preallocation, no vectorization.
Rows = 250;   % Number of rows in image matrix.
Rc = Rows/2;  % Where to place the image center (rows).
Cols = 250;   % Number of cols in image matrix.
Cc = Cols/2;  % Where to place the image center (cols).
Planes = 3;   % Three color planes
Rad = 50;     % Radius of circle to draw
myimage = zeros(Rows, Cols, Planes, 'uint8');
tic;
for r = 1:Rows
  for c = 1:Cols
    for p = 1:Planes
      if norm([r - Rc, c - Cc]) < Rad
        % Draw white circle
        myimage(r, c, p) = 255;
      end
    end
  end
end
t = toc;

figure
image(myimage)
fprintf('No vectorization, with preallocation.  Created matrix in %f seconds\n', t)
```



Loop over all points, color point white if it lies inside the circle of radius Rad.

Vectorized circle drawing program

```
clear all

% With vectorization and preallocation.
Rows = 250;   % Number of rows in image matrix.
Rc = Rows/2;  % Where to place the image center (rows).
Cols = 250;   % Number of cols in image matrix.
Cc = Cols/2;  % Where to place the image center (cols).
Planes = 3;   % Three color planes
Rad = 50;     % Radius of circle to draw
myimage = zeros(Rows, Cols, Planes, 'uint8');  % Create black background

tic;
% Create row and col matrices
[r, c] = meshgrid(1:Rows, 1:Cols);
% find all [r, c] inside circle and create idx matrix
idx = (r - Rc).*(r - Rc) + (c - Cc).*(c - Cc) < Rad*Rad;
% Now draw circle a new 2D matrix using idx
mat = zeros(Rows, Cols);
mat(idx) = 255;
% Now copy this matrix into the 3D array myimage
for p = 1:Planes
  myimage(:,:, p) = mat;
end
t = toc;

figure
image(myimage)
fprintf('Vectorization and preallocation.  Created matrix in %f seconds\n', t)
```

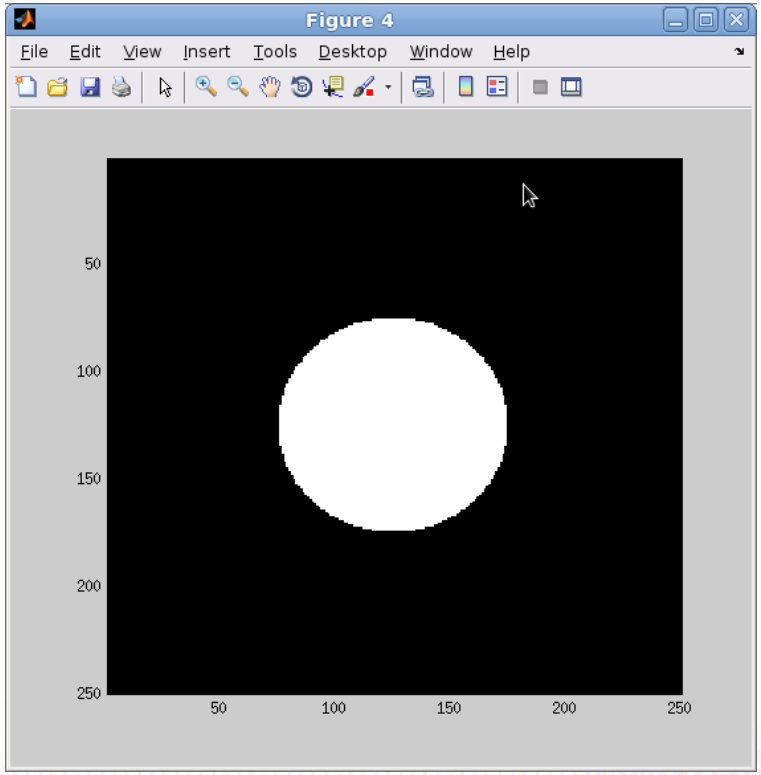
Use meshgrid to create plane of points. Then use logical indexing to select the points inside the circle.

Now color indexed points white.

Performance comparison

```
>> PreallocateArraysTypeVectorized
No vectorization, with preallocation.  Created matrix in 0.191326 seconds
Vectorization and preallocation.  Created matrix in 0.008339 seconds
```

- > 20x improvement using vectorized code – in Matlab.



Use “find”

```
>> A = randn(7)

A =

   -0.2725   -1.5771   -0.2991   -1.1564   -0.7982   -0.2938   -0.8655
    1.0984    0.5080    0.0229   -0.5336    1.0187   -0.8479   -0.1765
   -0.2779    0.2820   -0.2620   -2.0026   -0.1332   -1.1201    0.7914
    0.7015    0.0335   -1.7502    0.9642   -0.7145    2.5260   -1.3320
   -2.0518   -1.3337   -0.2857    0.5201    1.3514    1.6555   -2.3299
   -0.3538    1.1275   -0.8314   -0.0200   -0.2248    0.3075   -1.4491
   -0.8236    0.3502   -0.9792   -0.0348   -0.5890   -1.2571    0.3335

>> idx = find(abs(A)<0.1)

idx =

    11
    16
    27
    28

>> A(idx)

ans =

    0.0335
    0.0229
   -0.0200
   -0.0348
```

“find” returns a vector of indices

Another find example

```
A =

   -0.0942   -0.2883    1.0360
    0.3362    0.3501    2.4245
   -0.9047   -1.8359    0.9594

>> B = sqrt(A)

B =

   0.0000 + 0.3070i   0.0000 + 0.5369i   1.0178 + 0.0000i
   0.5798 + 0.0000i   0.5917 + 0.0000i   1.5571 + 0.0000i
   0.0000 + 0.9511i   0.0000 + 1.3549i   0.9795 + 0.0000i

>> idx = find(imag(B) ~= 0)

idx =

     1
     3
     4
     6

>> A(idx)

ans =

   -0.0942
   -0.9047
   -0.2883
   -1.8359
```

Exploit functions which return indicies

- Many functions return a value and an index.
- Min, max, intersect, union, sort, etc.

```
>> A = randn(1, 100);
>> [x, i] = min(A)

x =

   -2.3299


i =

    72

>> A(i)

ans =

   -2.3299
```

Index using a vector of indicies

```
>> y = randperm(20)

y =

    10    18    12    13     8    11    16    20     6    19     7     3    15     9
     5     4    14     1     2    17

>> [ys, idx] = sort(y)

ys =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14
    15    16    17    18    19    20


idx =

    18    19    12    16    15     9    11     5    14     1     6     3     4    17
    13     7    20     2    10     8

>> y(idx)

ans =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14
    15    16    17    18    19    20
```

Use dedicated matrix constructors

- repmat

```
>> A = [0 0 1; 0 1 0; 1 0 0]

A =

     0     0     1
     0     1     0
     1     0     0

>> B = repmat(A, 2, 2)

B =

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

Do the same thing using kron

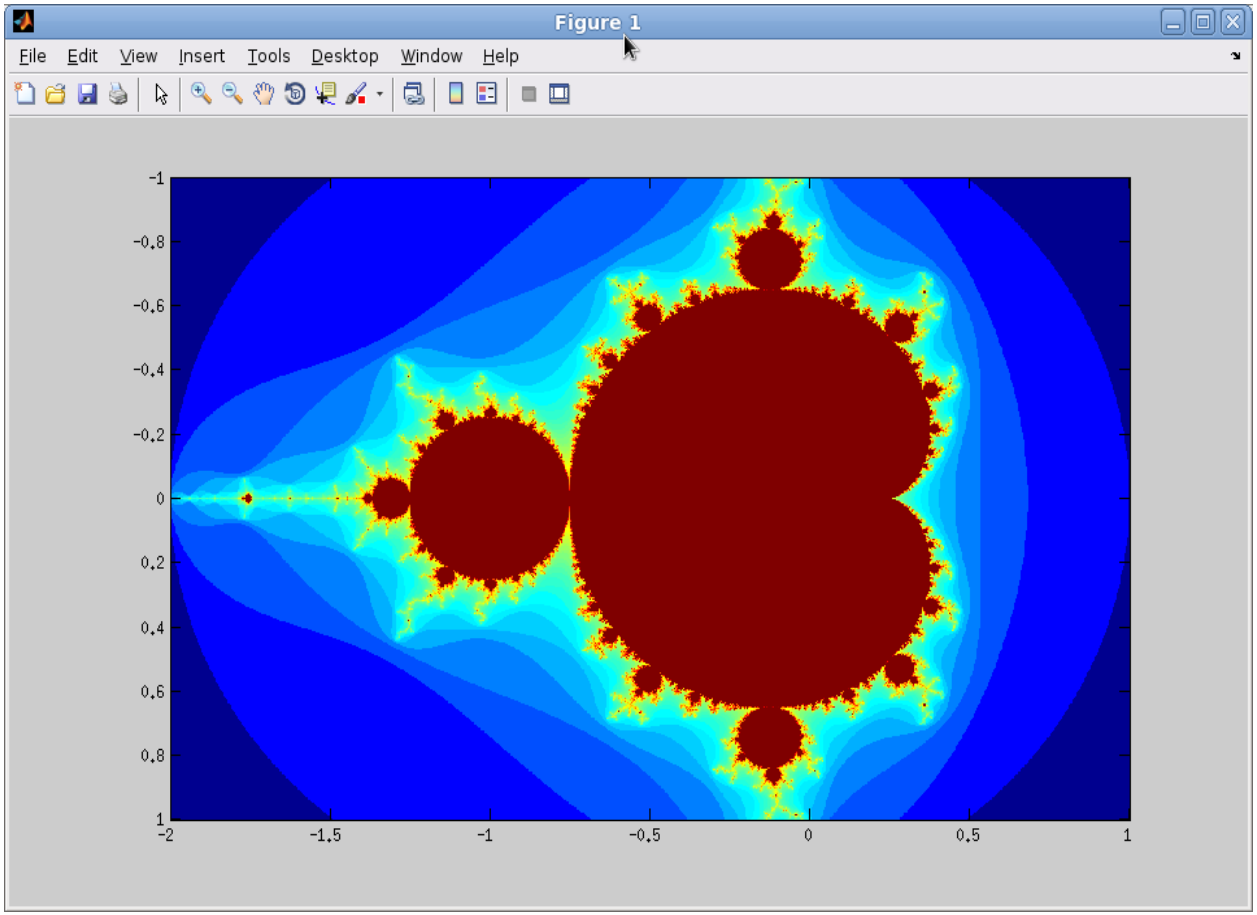
- kron

```
octave:4> A = [0 0 1; 0 1 0; 1 0 0]
A =

   0   0   1
   0   1   0
   1   0   0

octave:5> R = ones(2,2)
R =

   1   1
   1   1

octave:6> kron(R, A)
ans =

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

Profile your code with tic/toc

```
>> profile on
>> time_matmul
---------------  mymatmul  ---------------------
N =      3, avg multiplication time =       0.00054825 sec
N =      5, avg multiplication time =       0.00043700 sec
N =     10, avg multiplication time =       0.00324100 sec
N =     20, avg multiplication time =       0.02420525 sec
N =     30, avg multiplication time =       0.07175375 sec
N =     50, avg multiplication time =       0.30089525 sec
N =    100, avg multiplication time =       2.31823850 sec
N =    200, avg multiplication time =      21.46755650 sec
---------------  BLAS  ---------------------
N =      3, avg multiplication time =       0.01171150 sec
N =      5, avg multiplication time =       0.00001125 sec
N =     10, avg multiplication time =       0.00205325 sec
N =     20, avg multiplication time =       0.00002000 sec
N =     30, avg multiplication time =       0.00002800 sec
N =     50, avg multiplication time =       0.00008950 sec
N =    100, avg multiplication time =       0.00025350 sec
N =    200, avg multiplication time =       0.00083700 sec
My multiplication is O(3.004)
Matlabs multiplication is O(1.751)

ans =

     3     5    10    20    30    50   100   200

>> profile report
```

The Mandelbrot Set

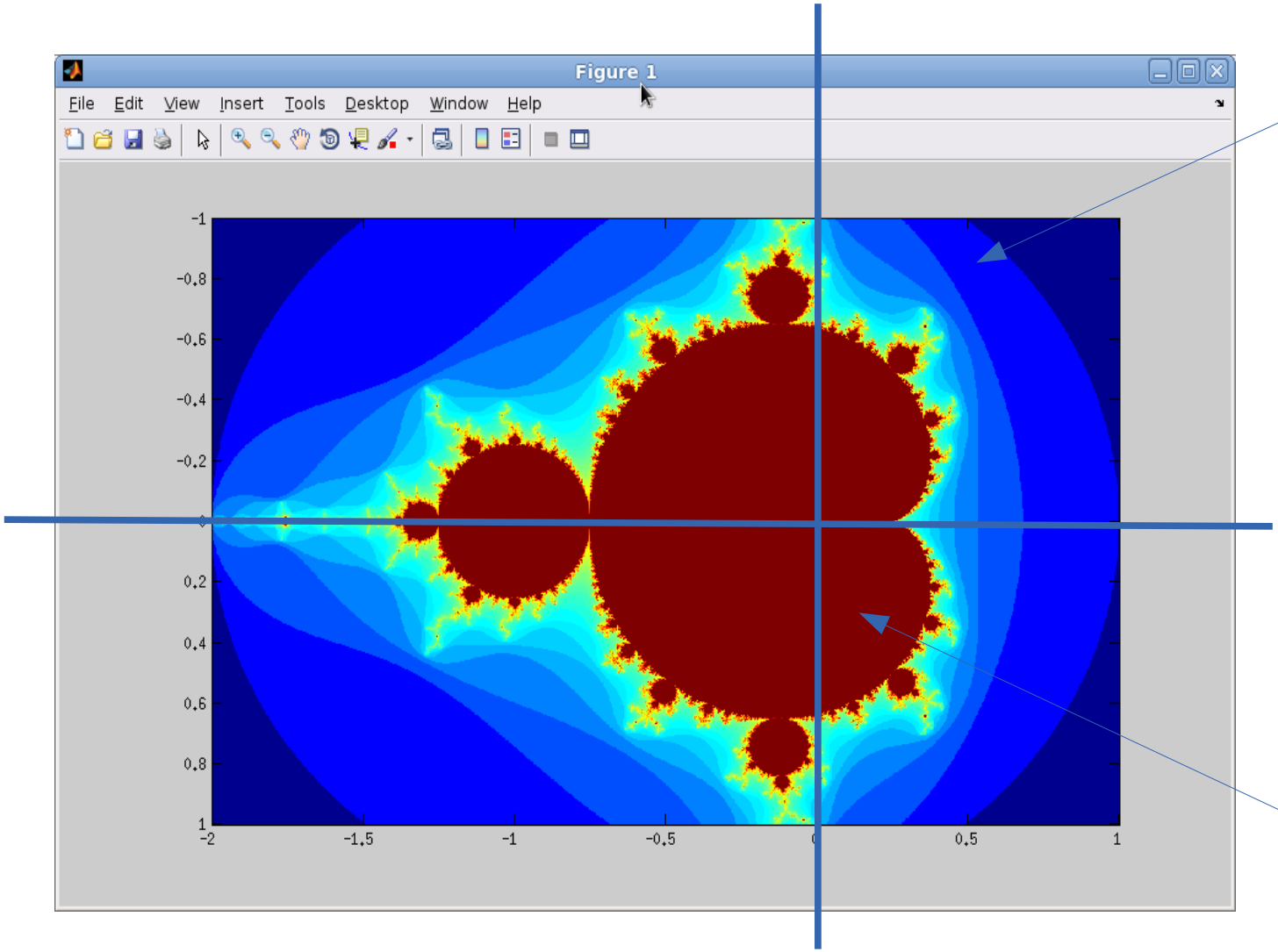


- Famous fractal.
- Fractal = Set with non-integer dimension.

Mandelbrot set mathematics

- Use iteration: $z_{n+1} = z_n^2 + c \qquad z_0 = 0$
- *z* and *c* are complex numbers
- This iteration will either diverge to infinity, or stay bounded as *c* is varied.
- Make plot in complex plane of c:
  - If iteration remains bounded, plot point dark red
  - If iteration diverges, plot point blue.
- Note that *z* diverges to infinity when $|z| \geq 2$

$$z_{n+1} = z_n^2 + c$$



z walks to infinity as equation is iterated

Iteration remains bounded

Algorithm

1. For cr = -2:.01:2
2. For ci = -2:.01:2
3. z = 0
4. For cnt = 1:max iterations
5. $z = z^2 + (cr + i*ci)$
6. If abs(z) > 2, iteration has diverged. Set pixel(cr, ci) blue, then break to next ci iteration.
7. End cnt loop
8. Color pixel(cr, ci) red
9. End ci loop
10. End cr loop

Example: Mandelbrot set
Non-vectorized

```
function Mandelbrot()
  % This computes and plots the Mandelbrot set using a non-vectorized
  % algorithm.

  t = tic();     % Start performance timer

  maxIterations = 255;  % When to decide point will not escape to inf.
  gridSize = 1000;      % This is length of each side of image matrix.

  % Domain over which to compute Mandelbrot set.
  xmin = -2;
  xmax = 1;
  ymin = -1;
  ymax = 1;

  % Start to setup z0 = x+iy values.
  x = linspace( xmin, xmax, gridSize );
  y = linspace( ymin, ymax, gridSize );

  % Preallocate matrix to hold result of iteration.
  % Note -- matrix rows are y, cols are x.
  c = zeros(length(y), length(x));
```

```
% Loop over all z = x+iy in complex plane
for xidx = 1:length(x)
  for yidx = 1:length(y)
    % Initialize z which will be iterated.
    z0 = complex(x(xidx), y(yidx));
    z = z0;
    for n = 0:maxIterations
      z = z*z + z0;
      if abs(z) <= 2
        % Still inside circle of radius 2
        c(yidx, xidx) = c(yidx, xidx) + 1;
      else
        % We've broken outside the circle -- stop iterating.
        break
      end
    end
    c(yidx, xidx) = log(c(yidx, xidx)+1);  % This enhances the colors plotted
  end
end
cpuTime = toc( t );  % Stop performance timer

% Show plot
imagesc( x, y, c );
fprintf('Elapsed computation time = %f sec\n', cpuTime)

end
```

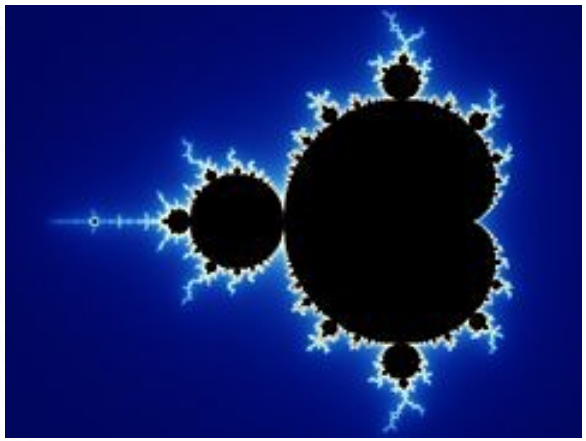
Iterate over points in complex plane

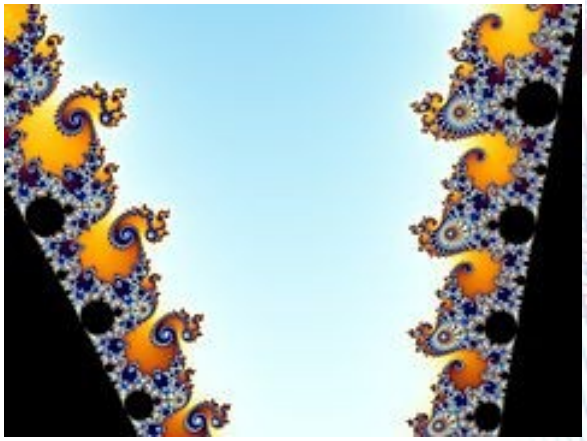
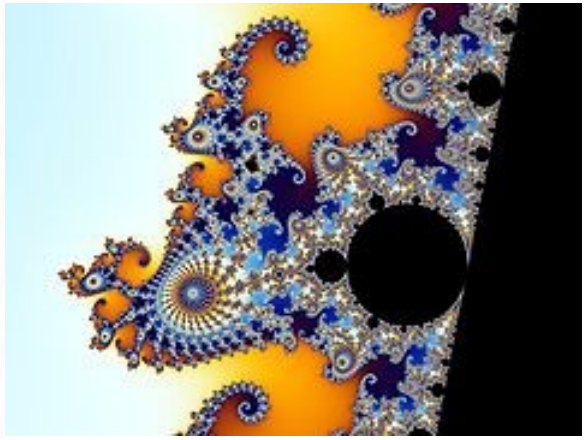
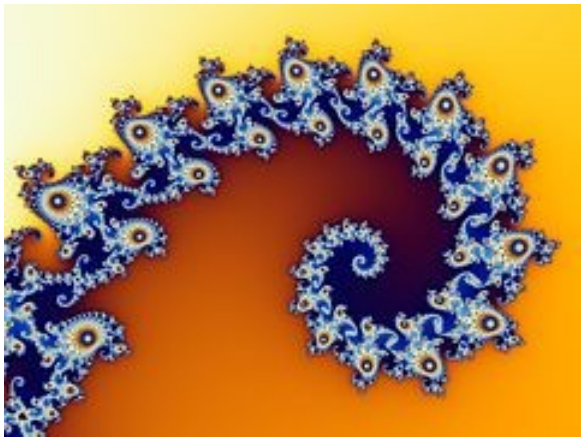
Iterate points to check divergence

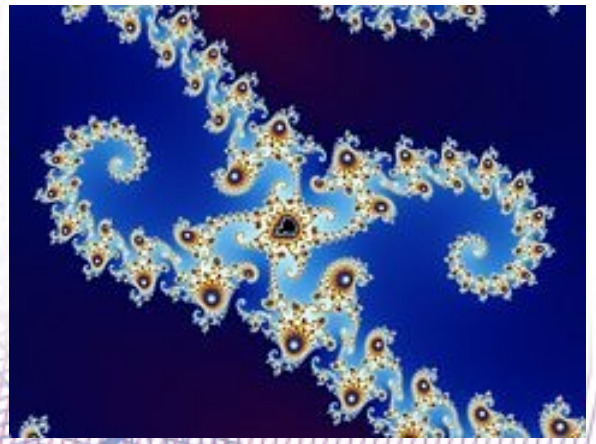
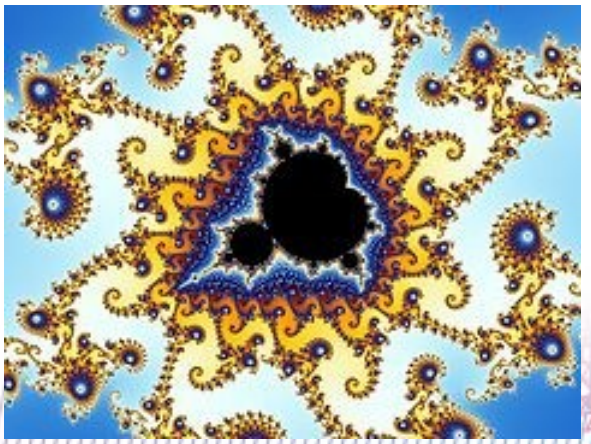
This version produces more colors than simple red/blue.

- Note three nested for loops.

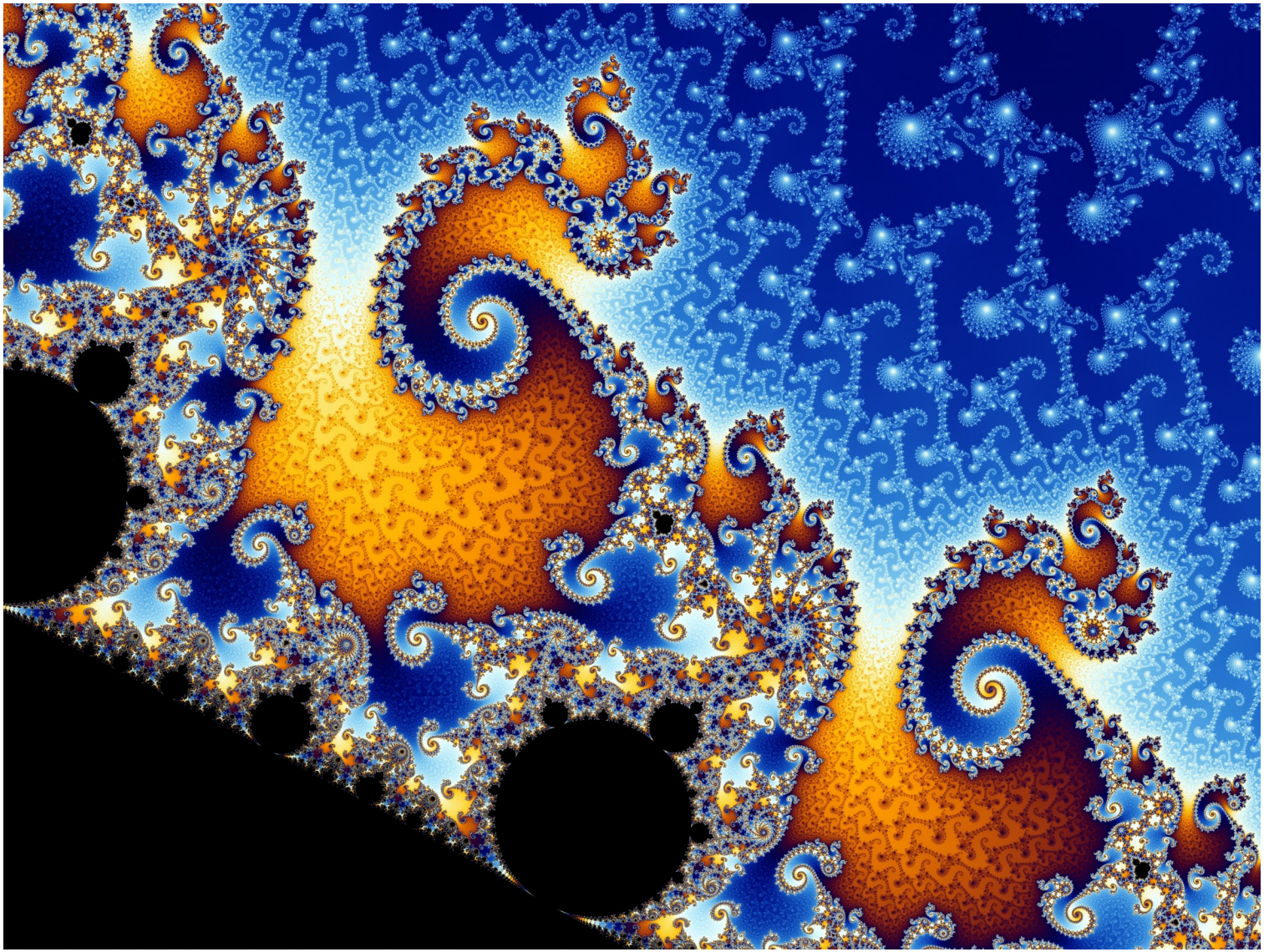
Zoom sequence



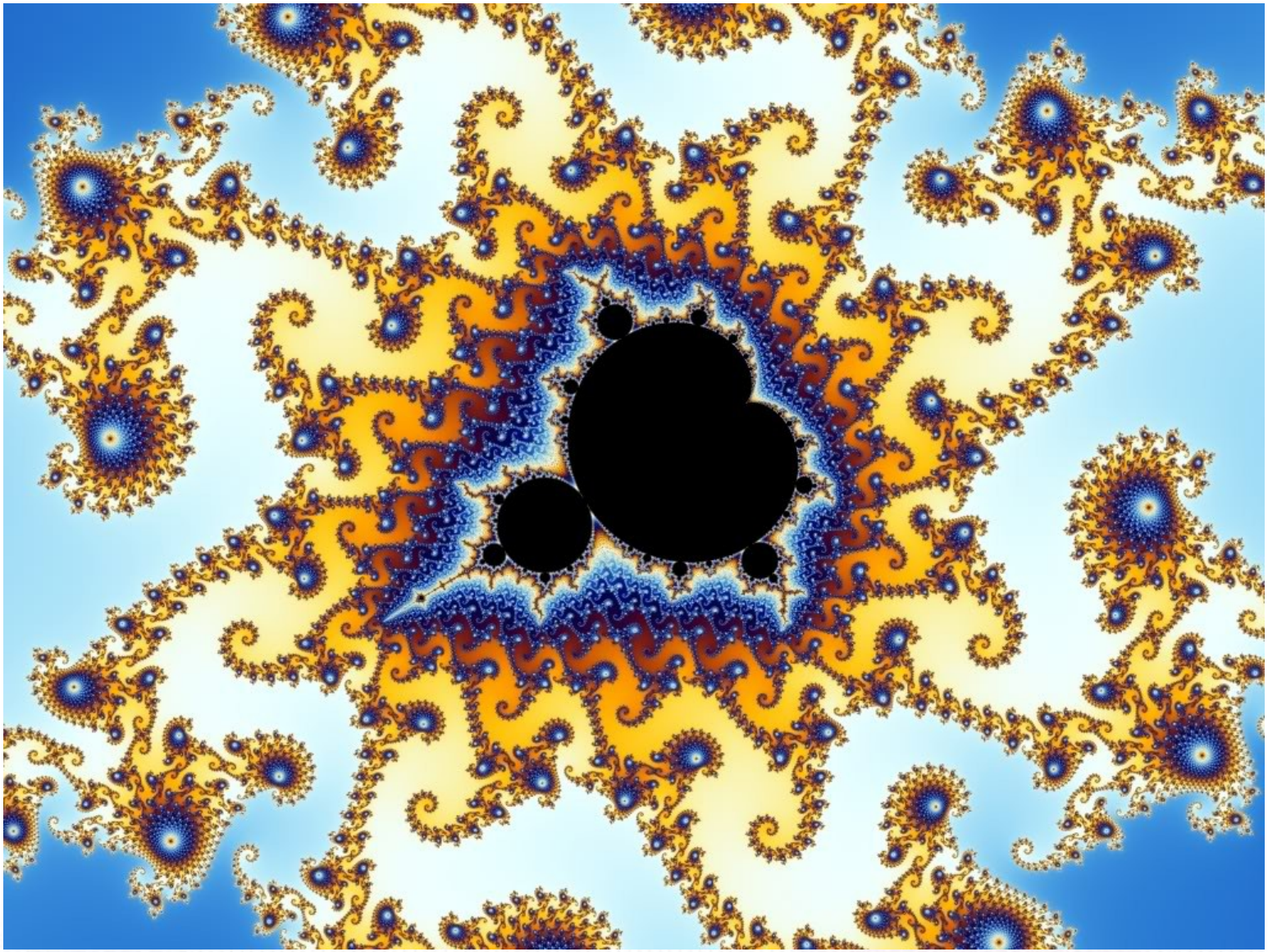




Zoom into Mandelbrot set



- Play with colors and iteration count...

Deep zoom into Mandelbrot set



Vectorized Mandelbrot code

```
function Mandelbrot_vectorized()
  % This computes and plots the Mandelbrot set using a fully-vectorized
  % algorithm.

  t = tic();     % Start performance timer

  maxIterations = 50;  % When to decide point will not escape to inf.
  gridSize = 1000;      % This is length of each side of image matrix.

  % Domain over which to compute Mandelbrot set.
  xmin = -2;
  xmax = 1;
  ymin = -1;
  ymax = 1;

  % Setup z0 = x+iy values.
  x = linspace( xmin, xmax, gridSize );
  y = linspace( ymin, ymax, gridSize );

  % Create initial z values
  [xGrid,yGrid] = meshgrid( x, y );
  z0 = complex(xGrid, yGrid);

  % Preallocate matrix to hold result of iteration.
  c = zeros( size(z0) );
```

Set up playing field using meshgrid

```
  % Calculate iteration
  z = z0;
  for n = 0:maxIterations
    z = z.*z + z0;
    % If the point is still inside the circle, add 1 to count.
    inside = (abs(z) <= 2);  % Find indicies of points inside circle
    c = c + inside;
  end
  c = log(c+1);  % Enhance colors plotted

  cpuTime = toc( t );   % Stop timer

  % Show plot
  imagesc( x, y, c );
  fprintf('Elapsed computation time = %f sec\n', cpuTime)

end
```

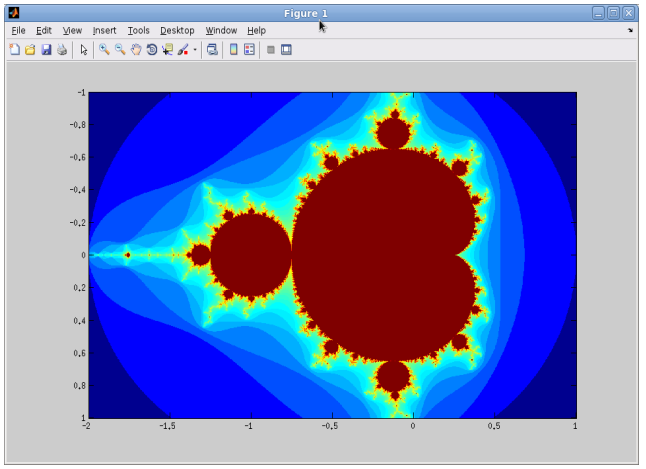
This statement updates entire complex plane at once

Use logical indexing to find points inside circle

Performance comparison

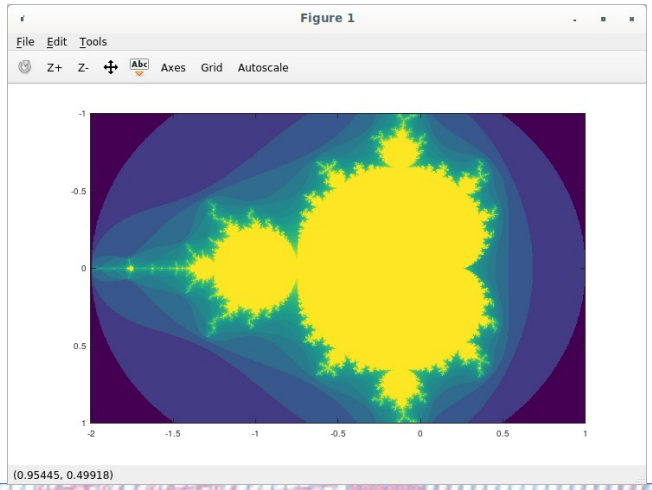
- Factor of 2 to 3 improvement for vectorized version (Matlab).

```
>> Mandelbrot
Elapsed computation time = 11.827231 sec
>> Mandelbrot_vectorized
Elapsed computation time = 4.307583 sec
```



- Factor of 150 improvement for vectorized version (Octave).

```
octave:9> Mandelbrot
Elapsed computation time = 253.655497 sec
octave:10> Mandelbrot_vectorized
Elapsed computation time = 1.591694 sec
```



Session summary

- Writing performant Matlab code
  - Preallocate arrays
  - Vectorize your code
  - Use specialized constructors for vectorization
  - Exploit functions which return index vectors
  - Use specialized matrix constructors like repmat, kron
- Mandelbrot set
- Mandelbrot set vectorized