If any of this comes across as clever, well prepared, or insightful it’s not my fault! I’m taking every thing from this excellent book.
Why Programming?

**Idealized computer.** "Please simulate the population dynamics of several species, some of which are infected with a disease, subject to Lotka Volterra equations with susceptible, infected, and recovered populations."

**Prepackaged software solutions.** Great, if it does exactly what you need.

**Computer programming.** Art of making a computer do what **you** want.

---

Ada Lovelace  
Analytic Engine
Languages

“Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.” – Donald Knuth

**Machine languages.** Tedious and error-prone.

**Natural languages.** Ambiguous and hard for computer to parse.

*Kids Make Nutritious Snacks.*
*Red Tape Holds Up New Bridge.*
*Police Squad Helps Dog Bite Victim.*
*Local High School Dropouts Cut in Half.*

[ real newspaper headlines, compiled by Rich Pattis ]

**High-level programming languages.** Acceptable tradeoff.
Why Matlab?

Matlab features.

- Widely used in scientific community.
- This campus pays for it.
- Many powerful abstractions relating to matrix manipulation, the cornerstone of scientific computing.
- Variety of automatic checks for mistakes in programs.
- Integrated development environment.
- Visualization facilities.
- Excellent documentation
- No strong types.

Cleve Moler
http://www.mathworks.com/company/aboutus/founders/clevemoler.html

“There are only two kinds of programming languages: those people always [gripe] about and those nobody uses.” – Bjarne Stroustrup
Why Matlab?

Matlab features.
- Matrix LABoratory
- Many useful scientific abstractions.
- Widely used.
- Widely available to those with money for such things.
- Variety of automatic checks for mistakes in programs.
- Well documented.

Caveat. No perfect language. Open source alternatives.

Our approach.
- Subset of Matlab: core Matlab and statistics tool box.

Develop general programming skills that are applicable to: C, C++, Java, Perl, Python, Ruby, Fortran, R, ... or whatever!
## A Subset of the Matlab Language

### Built-In Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int16</td>
<td>double</td>
</tr>
<tr>
<td>cell</td>
<td>struct</td>
</tr>
<tr>
<td>char</td>
<td>logical</td>
</tr>
</tbody>
</table>

### Flow Control

<table>
<thead>
<tr>
<th>Control</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>else</td>
</tr>
<tr>
<td>for</td>
<td>while</td>
</tr>
</tbody>
</table>

### Boolean

<table>
<thead>
<tr>
<th>Boolean</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td></td>
</tr>
</tbody>
</table>

### Input

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>load()</td>
<td></td>
</tr>
<tr>
<td>input()</td>
<td></td>
</tr>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>save()</td>
<td></td>
</tr>
<tr>
<td>fprintf()</td>
<td></td>
</tr>
</tbody>
</table>

### Math

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin()</td>
<td>cos()</td>
</tr>
<tr>
<td>log()</td>
<td>exp()</td>
</tr>
<tr>
<td>sqrt()</td>
<td>log10()</td>
</tr>
<tr>
<td>min()</td>
<td>max()</td>
</tr>
<tr>
<td>mod()</td>
<td>pi</td>
</tr>
</tbody>
</table>

### Numeric Operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>/</td>
<td>^</td>
</tr>
<tr>
<td>.*</td>
<td>&lt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;=</td>
</tr>
<tr>
<td>~=</td>
<td>[]</td>
</tr>
</tbody>
</table>

### String

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>'</td>
</tr>
<tr>
<td>size()</td>
<td>char()</td>
</tr>
<tr>
<td>strcmp()</td>
<td>strmatch()</td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Array</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = [1,2;3,4]</td>
<td>a(2,1)</td>
</tr>
<tr>
<td></td>
<td>size(a)</td>
</tr>
</tbody>
</table>

### Array Usage

<table>
<thead>
<tr>
<th>Usage</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(1,1) = 5</td>
<td>dot()</td>
</tr>
<tr>
<td>cross()</td>
<td>ones()</td>
</tr>
<tr>
<td>union()</td>
<td>rand()</td>
</tr>
<tr>
<td>imagesc()</td>
<td>A\b</td>
</tr>
</tbody>
</table>

### Punctuation

<table>
<thead>
<tr>
<th>Punctuation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>,</td>
<td>;</td>
</tr>
<tr>
<td>[</td>
<td>]</td>
</tr>
</tbody>
</table>
Features of Matlab

Enter commands here, and they are evaluated right away.

The current directory contents are here.

Here are all the variables in the environment.

This is a history of commands. Try dragging them back.
Using Matlab as a Calculator

Enter the following in to the Command Window and observe the results
- $4 \times 5$
- $\sin(\pi/2)$
- $x = 100$
- $y = 400$
- $x + y$
- $y = x^2$
- $\log(x)$
- $\log_{10}(x)$
- $x = \text{linspace}(0, 4\pi, 40)$
- $x = \text{linspace}(0, 4\pi, 40);$  
- $y = \cos(x);$  
- $\text{plot}(x, y)$
- $y_2 = x.^2;$  
- $\text{plot}(x, y_2, 'm')$
- $y_3 = x>2*\pi;$  
- $\text{plot}(x, y_3, '-.k', x, y, 'g');$

Consider the following questions
1. What does the semi-colon at the end do?
2. What are the arguments of linspace?
3. How do multiple data like $x = \text{linspace}(0, 1, 10)$ differ in the workspace from single data like $x = 1$?
4. How do multiple data differ from single data in terms of operations?
Programming in Matlab.

- **Create** the program by typing into the text editor, and save it as `HelloWorld.m`
- This is called a *script*. *Long, complex sets of instructions can be done this way.*

```plaintext
%*******************************************
% Prints "Hello, World"
% Everyone's first Matlab program.
%*******************************************

fprintf('Hello World\n');
```

`HelloWorld.m`
Programming in Matlab

Create the program by typing into the 'Command Window' `edit` HelloWorld.m

Execute it by:
- typing HelloWorld in the 'Command Window'.
- Hitting 'F5' on the keyboard (mouse must be in editor)
- Clicking the run icon
Built-in Data Types

**Data type.** A set of values and operations defined on those values.

Matlab does have data types, but by default everything is a matrix of doubles. Some other types follow.

<table>
<thead>
<tr>
<th>type</th>
<th>set of values</th>
<th>literal values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>characters</td>
<td>'A', '@'</td>
<td>compare</td>
</tr>
<tr>
<td>int16</td>
<td>16 bit integer</td>
<td>12</td>
<td>add (+), subtract(-), multiply(*), divide(/)</td>
</tr>
<tr>
<td>double</td>
<td>floating point numbers</td>
<td>17 12345</td>
<td>add (+), subtract(-), multiply(*), divide(/)</td>
</tr>
<tr>
<td>matrix</td>
<td>sets of numbers</td>
<td>[1 2 3 4]</td>
<td>add (+), subtract(-), multiply(.*), divide(/)</td>
</tr>
<tr>
<td>logical</td>
<td>truth values</td>
<td>true, false</td>
<td>and, or, not</td>
</tr>
</tbody>
</table>

To see how everything is a matrix, try this:

```matlab
a=1
a(1)
a(2)
a(2)=3
```
Trace

**Trace.** Table of variable values after each statement.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 1234;</td>
<td>1234</td>
<td>undefined</td>
<td>undefined</td>
</tr>
<tr>
<td>b = 99;</td>
<td>1234</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>t = a;</td>
<td>1234</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>a = b;</td>
<td>99</td>
<td>99</td>
<td>1234</td>
</tr>
<tr>
<td>b = t;</td>
<td>99</td>
<td>1234</td>
<td>1234</td>
</tr>
</tbody>
</table>
Concatenation: Subdivisions of a Ruler

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

ruler1 = ’1’;
ruler2 = [ruler1, ’2’, ruler1];
ruler3 = [ruler2, ’3’, ruler2];
ruler4 = [ruler3, ’4’, ruler3];
ruler4

‘1’
’1 2 1’
’1 2 1 3 1 2 1’

Does this concatenation method work with numbers? What type of data results?
Floating-Point Numbers

double data type. Useful in scientific applications.

<table>
<thead>
<tr>
<th>values</th>
<th>approximations to real numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>3.14159 6.022e23 -3.0 2.0 1.4142135623730951</td>
</tr>
<tr>
<td>operations</td>
<td>add subtract multiply divide</td>
</tr>
<tr>
<td>operators</td>
<td>+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.141 + .03</td>
<td>3.171</td>
</tr>
<tr>
<td>3.141 - .03</td>
<td>3.111</td>
</tr>
<tr>
<td>6.02e23 / 2</td>
<td>3.01e23</td>
</tr>
<tr>
<td>5.0 / 3.0</td>
<td>1.6666666666666667</td>
</tr>
<tr>
<td>1.0 / 0.0</td>
<td>Infinity</td>
</tr>
<tr>
<td>Math.sqrt(2.0)</td>
<td>1.4142135623730951</td>
</tr>
<tr>
<td>Math.sqrt(-1.0)</td>
<td>NaN</td>
</tr>
</tbody>
</table>
logical data type. Useful to control logic and flow of a program.

**Values** | true or false
---|---
**Literals** | true, false
**Operations** | and, or, not
**Operators** | &&, ||, ~

<table>
<thead>
<tr>
<th>a</th>
<th>~a</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

| a | b | a && b | a || b |
|---|---|---|---|
| false | false | false | false |
| false | true | false | true |
| true | false | false | true |
| true | true | true | true |

*Truth-table definitions of boolean operations*
**Comparisons.** Take operands of one type and produce an operand of type logical.

<table>
<thead>
<tr>
<th>op</th>
<th>meaning</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal</td>
<td><code>2 == 2</code></td>
<td><code>2 == 3</code></td>
</tr>
<tr>
<td><code>~=</code></td>
<td>not equal</td>
<td><code>3 ~= 2</code></td>
<td><code>2 ~= 2</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>2 &lt; 13</code></td>
<td><code>2 &lt; 2</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal</td>
<td><code>2 &lt;= 2</code></td>
<td><code>3 &lt;= 2</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>13 &gt; 2</code></td>
<td><code>2 &gt; 13</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal</td>
<td><code>3 &gt;= 2</code></td>
<td><code>2 &gt;= 3</code></td>
</tr>
</tbody>
</table>

- **non-negative discriminant?** \( (b*b - 4.0*a*c) \geq 0.0 \)
- **beginning of a century?** \( \text{mod(year,}100) = 0 \)
- **legal month?** \( (\text{month} \geq 1) \&\& (\text{month} \leq 12) \)
Exercise

Do the exercise labeled "Box Muller"
Conditionals and Loops
A Foundation for Programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math | text I/O

primitive data types | assignment statements

equivalent to a calculator
A Foundation for Programming

any program you might want to write

- objects
- functions and modules
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements

to infinity and beyond!
Control Flow

Control flow.

- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable us to choreograph control flow.

```
statement 1
statement 2
statement 3
statement 4

logical 1
true
false

statement 1
logical 2
true
false

statement 2
statement 3

straight-line control flow
control flow with conditionals and loops
```
If Statement

The if statement. A common branching structure.

- Check logical condition.
- If true, execute some statements.
- If false, execute other statements.

```
if boolean expression
    statement T;
else
    statement F;
end
```

can be any sequence of statements
The **if** statement. A common branching structure.

- **Check** boolean condition.
- **If** true, execute some statements.
- **If** false, execute other statements.

This works, can you see a problem with Matlab? Hint: try `max(rand(1,100))`
The While Loop
The while loop. A common repetition structure.

- Check a boolean expression.
- Execute a sequence of statements.
- Repeat.

while (logical expression)
    statement 1;
    statement 2;
end
While Loops: Powers of Two

**Ex.** Print first \( n \) powers of 2.
- Increment \( i \) from 1 to \( n \).
- Double \( v \) each time.

```
i = 0;
v = 1;
N = 6;

while (i <= N)
    disp(v);
    i = i + 1;
    v = 2 * v;
end
```

<table>
<thead>
<tr>
<th>i</th>
<th>v</th>
<th>i &lt;= N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>false</td>
</tr>
</tbody>
</table>

\( n = 6 \)
Q. Anything wrong with the following code for printing powers of 2?

```matlab
i = 0;
v = 1;
while (i <= N)
    disp(v);
    v = 2 * v;
end
```

Moment of panic. How to stop infinite loop? (Control C)
Exercise

Do the exercise labeled “Population Growth”
"A wonderful square root. Let’s hope it can be used for the good of mankind."

Copyright 2004, Sidney Harris, http://www.sciencecartoonsplus.com
While Loops: Square Root

Q. How might we implement $\text{sqrt}(\cdot)$?
A. To compute the square root of $c$:
   - Initialize $t_0 = c$.
   - Repeat until $t_i = c / t_i$, up to desired precision:
     set $t_{i+1}$ to be the average of $t_i$ and $c / t_i$.

\[
\begin{align*}
t_0 &= 2.0 \\
t_1 &= \frac{1}{2}(t_0 + \frac{2}{t_0}) = 1.5 \\
t_2 &= \frac{1}{2}(t_1 + \frac{2}{t_1}) = 1.4166666666666665 \\
t_3 &= \frac{1}{2}(t_2 + \frac{2}{t_2}) = 1.4142156862745097 \\
t_4 &= \frac{1}{2}(t_3 + \frac{2}{t_3}) = 1.4142135623746899 \\
t_5 &= \frac{1}{2}(t_4 + \frac{2}{t_4}) = 1.414213562373095
\end{align*}
\]

computing the square root of 2
Q. How might we implement $\sqrt{\text{ }}$?

A. To compute the square root of $c$:
   - Initialize $t_0 = c$.
   - Repeat until $t_i = c / t_i$, up to desired precision:
     - set $t_{i+1}$ to be the average of $t_i$ and $c / t_i$.

```matlab
c = 2; % Find the square root of this
t = c;
while (abs(t - c/t) > t*eps)
t = (c/t + t) / 2.0;
end
disp(t);
```

> sqrtnm
1.414213562373095

15 decimal digits of accuracy in 5 iterations
Newton-Raphson Method

Square root method explained.

- **Goal:** find root of function $f(x)$.
- Start with estimate $t_0$.
- Draw line tangent to curve at $x = t_i$.
- Set $t_{i+1}$ to be $x$-coordinate where line hits $x$-axis.
- Repeat until desired precision.

$$f(x) = x^2 - c$$ to compute $\sqrt{c}$ because Newton's method finds $x$ such that $x^2 - c = 0$

Hence, to find the zero, $t_{i+1} = t_i + \frac{f(x)}{f'(x)}$
Exercise

Do the exercise labeled “Newton’s Method”
The For Loop

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
For Loops

The for loop. Another common repetition structure.

- Execute initialization statement.
- Check boolean expression.
- Execute sequence of statements.
- Execute increment statement.
- Repeat.

```
for counter = start:step:finish
  statement 1;
  statement 2;
end %for
```

Diagram:

- Initialize `counter`.
- Check boolean expression.
  - If true, execute:
    - Increment `counter`.
    - Execute `statement 1`.
    - Execute `statement 2`.
    - Repeat.
  - If false, stop.

loop continuation condition, `counter <= finish`
For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize \texttt{ruler} to empty string.
- For each value \texttt{i} from 1 to \texttt{N}:
  - sandwich two copies of \texttt{ruler} on either side of \texttt{i}.

\begin{verbatim}
N = 10;
ruler = ''; 
for ii=1:N
    ruler = [ruler,num2str(ii),ruler];
end
ruler
\end{verbatim}

Try \texttt{N=100}! What is this happening?
Control Flow Summary

**Control flow.**
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enables us to choreograph the control flow.

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line programs</td>
<td>All statements are executed in the order given.</td>
<td></td>
</tr>
<tr>
<td><strong>Conditionals</strong></td>
<td>Certain statements are executed depending on the values of certain variables.</td>
<td>if elseif if-else</td>
</tr>
<tr>
<td><strong>Loops</strong></td>
<td>Certain statements are executed repeatedly until certain conditions are met.</td>
<td>while for</td>
</tr>
</tbody>
</table>
Nesting
**Nested If Statements**

**Ex.** Pay a certain tax rate depending on income level.

```plaintext
if (income < 47450) rate = 0.22;
    elseif (income < 114650) rate = 0.25;
    elseif (income < 174700) rate = 0.28;
    elseif (income < 311950) rate = 0.33;
else
    rate = .35
end

graduated income tax calculation
```

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

5 mutually exclusive alternatives
Q. Anything wrong with the following for income tax calculation?

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

rate = 0.35;
if (income < 47450) rate = 0.22; end;
if (income < 114650) rate = 0.25; end;
if (income < 174700) rate = 0.28; end;
if (income < 311950) rate = 0.33; end;

wrong graduated income tax calculation
Arrays

This lecture. Store and manipulate huge quantities of data.

Array. Indexed sequence of values of the same type.

Examples.

- 52 playing cards in a deck.
- 11 thousand undergrads at UM.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 trillion cells in the human body.
- \(6.02 \times 10^{23}\) particles in a mole.
Many Variables of the Same Type

**Goal.** 10 variables of the same type.

```matlab
% tedious and error-prone

a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a9 = 0.0;
a9 = 0.0;

x = a4 + a8;
```
Arrays in Matlab

Matlab has special language support for arrays.

- To make an array, simply create it.
- To access element $i$ of array named $a$, use $a(i)$.
- Array indices start at 1.
- Matlab provides many ways to create arrays. For instance:

```matlab
N=100;
a = 1:N;   % sequence of integers
a = 0:2:N; % sequence of even integers
a = linspace(0,1,N); % 0 to 1, N subdivisions
a = zeros(N); % array of all zeros
a = ones(N); % array of all ones
a = rand(N); % array of random numbers [0,1]
```
Setting 2D Array Values in a Script

Initialize 2D array by listing values.

\[
p = [\begin{array}{ccccc}
0.02 & 0.92 & 0.02 & 0.02 & 0.02 \\
0.02 & 0.02 & 0.32 & 0.32 & 0.32 \\
0.02 & 0.02 & 0.02 & 0.92 & 0.02 \\
0.92 & 0.02 & 0.02 & 0.02 & 0.02 \\
0.47 & 0.02 & 0.47 & 0.02 & 0.02 \\
\end{array}];
\]
**Dot product.** Given two vectors \( x[] \) and \( y[] \) of length \( N \), their dot product is the sum of the products of their corresponding components.

\[
\begin{align*}
x &= [0.3, 0.6, 0.1]; \\
y &= [0.5, 0.1, 0.4]; \\
dp &= x*y \\
ep &= x.*y
\end{align*}
\]
Self-Avoiding Walk

Model.

- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.

Applications. Polymers, statistical mechanics, etc.

Q. What fraction of time will you escape in an 5-by-5 lattice?
Q. In an N-by-N lattice?
Q. In an N-by-N-by-N lattice?
clear; % Clear all variables, good practice to reduce side effects
N = 25; % lattice size
T = 100; % number of trials
deadEnds = 0; % trials resulting in dead end

for t=1:T
    a = logical(zeros(N,N)); % intersections visited
    x = floor(N/2); y = floor(N/2); % start in center position
    while (x > 1 && x < N && y > 1 && y < N)
        if (a(x-1,y) && a(x+1,y) && a(x,y-1) && a(x,y+1))
            deadEnds = deadEnds + 1;
            imagesc(a); % Output plot if dead end
            drawnow; % Needed if plotting inside a loop (animation)
            break;
        end

        a(x,y) = true; % mark as visited

        r = rand();
        if (r < 0.25) if (~a(x+1,y)) x=x+1; end %if
        elseif (r < 0.50) if (~a(x-1,y)) x=x-1; end %if
        elseif (r < 0.75) if (~a(x,y+1)) y=y+1; end %if
        elseif (r < 1.00) if (~a(x,y-1)) y=y-1; end %if
        end %outer if
    end %while not escaped
end %for t

fprintf('%6.2f percent dead ends.
',100*deadEnds/T);

move or remove these plotting lines if you like, or need to speed it up.
Self-Avoiding Walks
Exercise

Do the exercise labeled “Self-avoiding random walk”