Lecture 4
Representing Data on the Computer

Ramani Duraiswami
AMSC/CMSC 662
Fall 2009
• $x = \pm (1+f) \times 2^e$
• $0 \cdot f < 1$
• $f = (\text{integer} < 2^{52})/2^{52}$
• $-1022 \leq e \leq 1023$
• $e = \text{integer}$
Effects of floating point

Finite $f$ implies finite precision.

Finite $e$ implies finite range

Floating point numbers have discrete spacing, a maximum and a minimum.
Effects of floating point

- $\text{eps}$ is the distance from 1 to the next larger floating-point number.
- $\text{eps} = 2^{-52}$
- In Matlab

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>eps</td>
<td>$2^{(-52)}$</td>
<td>$2.2204\times10^{-16}$</td>
</tr>
<tr>
<td>realmin</td>
<td>$2^{(-1022)}$</td>
<td>$2.2251\times10^{-308}$</td>
</tr>
<tr>
<td>realmax</td>
<td>$(2-\text{eps})\times2^{1023}$</td>
<td>$1.7977\times10^{308}$</td>
</tr>
</tbody>
</table>
Rounding vs. Chopping

• **Chopping**: Store $x$ as $c$, where $|c| < |x|$ and no machine number lies between $c$ and $x$.

• **Rounding**: Store $x$ as $r$, where $r$ is the machine number closest to $x$.

• **IEEE standard arithmetic uses rounding**.
Machine Epsilon

- **Machine epsilon** is defined to be the smallest positive number which, when added to 1, gives a number different from 1.
  - Alternate definition (1/2 this number)

- **Note:** Machine epsilon depends on d and on whether rounding or chopping is done, but does not depend on m or M!
Some numbers cannot be exactly represented.

\[
\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \frac{1}{2^{12}} + \ldots
\]

\[
t = (1 + \frac{9}{16} + \frac{9}{16^2} + \frac{9}{16^3} + \ldots + \frac{9}{16^{12}} + \frac{10}{16^{13}}) \cdot 2^{-4}
\]
\[ x = 1; \text{while } 1+x > 1, \ x = x/2, \ \text{pause}(.02), \ \text{end} \]

\[ x = 1; \text{while } x+x > x, \ x = 2*x, \ \text{pause}(.02), \ \text{end} \]

\[ x = 1; \text{while } x+x > x, \ x = x/2, \ \text{pause}(.02), \ \text{end} \]
Floating point operations

- Basic arithmetic operations
  - addition, subtraction, multiplication, and division (and sometimes the square root).

- The IEEE standard specifies that these operations must return the correctly rounded result (provided they are within the range of normalized floating-point numbers).

- \((a \circ b) = (a \circ b)(1 + \varepsilon)\);

- Order of computations becomes important

- sum1 = \(a + (b + c)\) \quad sum2 = (a + b) + c
  - sum1 and sum2 may not have the same value.
  - 472635.0000 + 27.5013 - 472630.0000 = 32.5013
  - If we compute this sum in the order given in six digit arithmetic, we get
    - 472635.0000 + 27.5013 = 472663.0000
    - 472663 - 472630 = 33
    - which is accurate to only two digits. On the other hand
    - 472635 - 472630 = 5: 5.0000 + 27.5013 = 32.5013
Errors can be magnified

- Errors can be magnified during computation.
- Let us assume numbers are known to 0.05% accuracy.
- Example: $2.003 \times 10^0$ and $2.000 \times 10^0$
  - both known to within $\pm .001$
- Perform a subtraction. Result of subtraction: $0.003 \times 10^0$
- but true answer could be as small as $2.002 - 2.001 = 0.001$, or as large as $2.004 - 1.999 = 0.005$!

- Absolute error of 0.002
- Relative error of 200% !
- Adding or subtracting causes the bounds on absolute errors to be added
Error effect on multiplication/division

• Let $x$ and $y$ be true values
• Let $X=x(1+r)$ and $Y=y(1+s)$ be the known approximations
• Relative errors are $r$ and $s$
• What is the errors in multiplying the numbers?
• $XY=xy(1+r)(1+s)$
• Absolute error $=|xy(1-rs-r-s-1)| = (rs+r+s)xy$
• Relative error $=|(xy-XY)/xy|$
  
  $$= |rs+r+s| \leq |r| + |s| + |rs|$$
• If $r$ and $s$ are small we can ignore $|rs|$
• Multiplying/dividing causes relative error bounds to add
Effects of floating point errors

- Singular equations will only be nearly singular
- Severe cancellation errors can occur

```matlab
x = 0.988:.0001:1.012;
y = x.^7-7*x.^6+21*x.^5-35*x.^4+35*x.^3-21*x.^2+7*x-1;
plot(x,y)
```

```matlab
17x_1 + 5x_2 = 22
1.7x_1 + 0.5x_2 = 2.2
A = [17 5; 1.7 0.5]
b = [22; 2.2]
x = A\b
```

```
x =
-1.0588
  8.0000
```
Measuring error

• **Absolute error** in $c$ as an approximation to $x$:
  $$|x - c|$$

• **Relative error** in $c$ as an approximation to nonzero $x$:
  $$|(x - c)/x|$$
Floating point exceptions

Exceptions set a flag that can be queried by the programmer

Some are trapped and the system can be set to abort the process

Operations involving Infs and NaNs have a certain logic, and essentially continue to produce them

Overflow leads to Infs

0/0 inf/inf, sqrt of negative number etc. lead to NaN

Underflow is a non fatal exception

IEEE requires gradual underflow

<table>
<thead>
<tr>
<th>$k$</th>
<th>representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>$3.14159 \cdot 10^{-98}$</td>
</tr>
<tr>
<td>99</td>
<td>$3.14159 \cdot 10^{-99}$</td>
</tr>
<tr>
<td>100</td>
<td>$0.31416 \cdot 10^{-99}$</td>
</tr>
<tr>
<td>101</td>
<td>$0.03142 \cdot 10^{-99}$</td>
</tr>
<tr>
<td>102</td>
<td>$0.00314 \cdot 10^{-99}$</td>
</tr>
<tr>
<td>103</td>
<td>$0.00031 \cdot 10^{-99}$</td>
</tr>
<tr>
<td>104</td>
<td>$0.00003 \cdot 10^{-99}$</td>
</tr>
<tr>
<td>105</td>
<td>$0.00000 \cdot 10^{-99}$</td>
</tr>
</tbody>
</table>
Bit operations

- Bitwise logical operations
  - combine corresponding bits of two words to give another word.
  - operations: **and**, **or**, **xor** (exclusive or), complement

- Example
  - 0011 or 0101 = 0111
  - 0011 and 0101 = 0001
  - 0011 xor 0101 = 0110
  - 0011 complement = 1100

- Bitwise shift operations come in two flavors: logical and arithmetic

- Logical shifts simply shift the bits and replace spaces with zeros
  - 10110110 shifted right by three bits is 00010110.
  - Shifted left by three bits it becomes 10110000.
  - Bits that are shifted out are lost.
  - Main use is quick multiplication and division by 2

- The main use for shifts: **quickly** multiply and divide by powers of 2 of integers
  - multiplying by 00101 by 2 amounts to doing a left shift to 01010
  - multiplying by 4 amounts to doing two left shifts to 10100

- If numbers are too large, multiplication doesn’t produce valid results
  - e.g., 10000000 (128d) cannot be left-shifted to obtain 256 using 8-bit values

- Similarly, dividing by powers of two amounts to doing right shifts:
  - right shifting 10010 (18d) leads to 01001 (9d)
Other shifts

• Arithmetic shift
  – Give the same quick instruction level multiplication ability to signed numbers
  – (note that signed integers are stored in 2’s complement notation, so you will have to understand how they work)

• Rotate shifts
  – Bits that move out from the right (or left) reappear on the left (or right)
Character Representations

- ASCII – PC workstations
- EBCDIC – IBM Mainframes
- Unicode – International Character sets
ASCII

- Original ASCII: American Standard Code for Information Interchange
  7-bit coded character set for information interchange
- Specifies coding of space and a set of 94 characters (letters, digits and punctuation or mathematical symbols) suitable for the interchange of basic English language documents.
- Extended ASCII: 8 bit characters
  - All western European languages
- Groups of characters are called a string
- Several functions in programming languages to manipulate strings
### 7 – bit ASCII Code Set

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Comment</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Comment</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>NUL</td>
<td>Null</td>
<td>32</td>
<td>20</td>
<td>Space</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>SOH</td>
<td>Start of Heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>Exclamation</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>STX</td>
<td>Start of Text</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>Quotation Mark</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>ETX</td>
<td>End of Text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>Hash</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>EOT</td>
<td>End of Transmission</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>Dollar</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>ENQ</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>Percent</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>ACK</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>Ampersand</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>BEL</td>
<td>Bell (Ding!)</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>Single Quote</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>BS</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>Left Parenthesis</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>HT</td>
<td>Horizontal Tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>Right Parenthesis</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>LF</td>
<td>Line Feed</td>
<td>42</td>
<td>2A</td>
<td>*</td>
<td>Asterisk</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>VT</td>
<td>Vertical Tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>Plus</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>FF</td>
<td>Form Feed (new page)</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>Comma</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>CR</td>
<td>Carriage Return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>Hyphen</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>SO</td>
<td>Shift Out</td>
<td>46</td>
<td>2E</td>
<td></td>
<td></td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>SI</td>
<td>Shift In</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>Slash</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>DLE</td>
<td>Data Link Escape</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>Null Character</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>DC1</td>
<td>Device Control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>Number 1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>DC2</td>
<td>Device Control 2</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>Number 2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>DC3</td>
<td>Device Control 3</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>Number 3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>DC4</td>
<td>Device Control 4</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>Number 4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>NAK</td>
<td>Negative Acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>Number 5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>SYN</td>
<td>Synchronous Idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>Number 6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>EBT</td>
<td>End of Transmission Block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>Number 7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>CAN</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>Number 8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>EM</td>
<td>End of Medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>Number 9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>SUB</td>
<td>Substitute</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>Colon</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>ESC</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>Semicolon</td>
<td>91</td>
<td>5B</td>
<td>]</td>
<td>]</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>FS</td>
<td>File Separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>Less Than</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>GS</td>
<td>Group Separator</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>Equal</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>RS</td>
<td>Record Separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>Greater Than</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>US</td>
<td>Unit Separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>Question Mark</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96</td>
<td>60</td>
<td>a</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97</td>
<td>61</td>
<td>b</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98</td>
<td>62</td>
<td>c</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99</td>
<td>63</td>
<td>d</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>64</td>
<td>e</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101</td>
<td>65</td>
<td>f</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>102</td>
<td>66</td>
<td>g</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td>67</td>
<td>h</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>104</td>
<td>68</td>
<td>i</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>105</td>
<td>69</td>
<td>j</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>106</td>
<td>6A</td>
<td>k</td>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>107</td>
<td>6B</td>
<td>l</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>108</td>
<td>6C</td>
<td>m</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>109</td>
<td>6D</td>
<td>n</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>110</td>
<td>6E</td>
<td>o</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
<td>6F</td>
<td>p</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112</td>
<td>70</td>
<td>q</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>113</td>
<td>71</td>
<td>r</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>114</td>
<td>72</td>
<td>s</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>115</td>
<td>73</td>
<td>t</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>116</td>
<td>74</td>
<td>u</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117</td>
<td>75</td>
<td>v</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>118</td>
<td>76</td>
<td>w</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>119</td>
<td>77</td>
<td>x</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120</td>
<td>78</td>
<td>y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>121</td>
<td>79</td>
<td>z</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>122</td>
<td>7A</td>
<td>{</td>
<td>Left Curly Bracket</td>
<td>123</td>
<td>7B</td>
<td>}</td>
<td>Right Curly Bracket</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>124</td>
<td>7C</td>
<td>l</td>
<td>Left Angle Bracket</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>125</td>
<td>7D</td>
<td>}</td>
<td>Right Angle Bracket</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>126</td>
<td>7E</td>
<td>~</td>
<td>Tilde</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>127</td>
<td>7F</td>
<td>DEL</td>
<td>Delete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unicode

- Two byte character set to represent all of the world's characters in modern computer use,
- including technical symbols and special characters used in publishing.
- separate values for up to 65,536 characters. Unicode-enabled functions are often referred to as "wide-character" functions.
Performance

• One way to compare algorithms that solve the same problem is to count the number of floating-point operations they perform.

• Although such counts underestimate the execution time, that time is frequently proportional to the count.

• Count depends on the size of the problem (the order) and a constant.

• In comparing two algorithms of the same order, one must examine the order constant.

• For constants of different order, the one with the higher order will ultimately be faster.
  – But ultimately may never come in practice.
Matrix-vector product

- Matrix-vector multiplication applies a linear transformation to a vector:

\[
M \cdot v = \begin{bmatrix}
M_{11} & M_{12} & M_{13} \\
M_{21} & M_{22} & M_{23} \\
M_{31} & M_{32} & M_{33}
\end{bmatrix} \begin{bmatrix}
v_x \\
v_y \\
v_z
\end{bmatrix}
\]

- How many operations does a matrix vector product take?
matrix vector product

- Access matrix
  - Element-by-element along rows
  - Element-by-element along columns
- In either case there are two loops
- Inner loop has one multiply and add
- Done N times
- Outer loop is done M times

```matlab
[m,n]=size(A);
y = zeros(m,1);
for i=1:m,
    for j=1:n,
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

```matlab
[m,n]=size(A);
y = zeros(m,1);
for i=1:m,
    y(i) = A(i,:) * x;
end
```

```matlab
[m,n]=size(A);
y = zeros(m,1);
for j=1:n,
    y = y + A(:,j)*x(j);
end
```
Asymptotic Equivalence

- \( f(n) \sim g(n) \)

\[
\lim_{n \to \infty} \left( \frac{f(n)}{g(n)} \right) = 1
\]
Little Oh

• **Asymptotically smaller:**

\[ f(n) = o(g(n)) \]

\[
\lim_{n \to \infty} \left( \frac{f(n)}{g(n)} \right) = 0
\]
Big Oh

• Asymptotic Order of Growth:
  • \( f(n) = O(g(n)) \)

\[
\limsup_{n \to \infty} \left( \frac{f(n)}{g(n)} \right) < \infty
\]
The Oh’s

If $f = o(g)$ or $f \sim g$ then $f = O(g)$

$\lim = 0 \quad \lim = 1 \quad \lim < \infty$
The Oh’s

If $f = o(g)$, then $g \neq O(f)$

$$\lim_{x \to \infty} \frac{f}{g} = 0 \quad \quad \lim_{x \to \infty} \frac{g}{f} = \infty$$
Big Oh

• Equivalently,
  
  • \( f(n) = O(g(n)) \)

\[ \exists c, n_0 \geq 0 \ \forall n \geq n_0 \quad |f(n)| \leq c \cdot g(n) \]
Big Oh

\[ f(x) = O(g(x)) \]
Computer Performance

• Complexity
  – Time
  – Memory
  – Communication
  – Notation

• Statistical measurement
Amdahl's law

- When a task can be divided into independent subtasks, speeding up one of the subtasks will speed up the original task.
- Amdahl's law is a formula that shows the limits of speeding up a subtask and suggests which one to work on. In its form it also embodies the law of diminishing returns.
- E.g.: Two subtasks A and B
- \( T = T_A + T_B \)
- How much do we gain by altering task B to reduce the time \( T_B \)?
- Let \( T \) represents the time to execute a program, \( T_A \) is the CPU time spent actually spent executing the program, and \( T_B \) is the time when the CPU is idle waiting for input.
- To make the notion of 'gain' precise, let \( T_\sigma \) be reduced to \( T_\sigma / \sigma \), where \( \sigma > 1 \). Then the new total time is

\[
T_\sigma = T_A + \sigma^{-1}T_B.
\]

Speedup is defined as

\[
S(\sigma) = \frac{T}{T_\sigma} = \frac{T_A + T_B}{T_A + \sigma^{-1}T_B}.
\]
Diminishing returns

\[ f_A = \frac{T_A}{T} \quad \text{and} \quad f_B = \frac{T_B}{T}. \]

Thus \( f_A \) is the fraction of the total time accounted for by task A, and \( f_B \) is the fraction of the total time accounted for by task B. These quantities are not independent but satisfy the relation

\[ f_A + f_B = 1. \]

\[ S(\sigma) = \frac{1}{\frac{1}{f_A} + \sigma^{-1}f_B}. \]

Then as \( \sigma \to \infty \), the speedup \( S(\sigma) \)

\[ S(\infty) = \frac{1}{f_A}. \]