2072U Computational Science I
Winter 2022

Week	Topic
1	Introduction
1–2	Solving nonlinear equations in one variable
3–4	Solving systems of (non)linear equations
5–6	Computational complexity
6–8	Interpolation and least squares
8–10	Integration & differentiation
10–12	Additional Topics

1. Complexity of algorithms
2. Big-Oh notation

Key questions:

- What is the computational complexity of
  - Gaussion elimination / LU-decomposition?
  - polynomial evaluation?
- What does Landau notation (“Big-Oh”) mean?
- How can the implications of Landau notation be visualised?
- How does complexity relate to actual performance of programs?

Summary of what we learned so far:

Summary of what we learned so far:

- *Computational complexity* is counted in the number of *floating-point operations* (FLOPS).

Summary of what we learned so far:

- *Computational complexity* is counted in the number of *floating-point operations* (FLOPS).
- Usually, the (maximal) number of flops in a code can be found as
  - a sum for (nested) loops or
  - the solution to a recursion relation for recursive codes.

Summary of what we learned so far:

- *Computational complexity* is counted in the number of *floating-point operations* (FLOPS).
- Usually, the (maximal) number of flops in a code can be found as
  - a sum for (nested) loops or
  - the solution to a recursion relation for recursive codes.
- Under a number of simplifying assumptions, the number of flops determines the time it takes to run a code.

Summary of what we learned so far:

- *Computational complexity* is counted in the number of *floating-point operations* (FLOPS).
- Usually, the (maximal) number of flops in a code can be found as
  - a sum for (nested) loops or
  - the solution to a recursion relation for recursive codes.
- Under a number of simplifying assumptions, the number of flops determines the time it takes to run a code.
- The number of flops taken for some simple computations:

sum of n terms	$n-1$
product of n factors	$n-1$
dot product of n-vectors	$2n-1$
$n \times n$ matrix–vector product	$2n^2-n$
$n \times n$ matrix–matrix product	$2n^3-n^2$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$
    end
  end
end

So the number of flops is:

\# flops$=$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

flops$=$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\#\text{ flops} = \sum_{i=1}^{n-1}$$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\# \text{ flops} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n}$$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\#\text{ flops} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Big(1 +$$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

flops$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Bigg(1 + \sum_{k=i+1}^{n+1}$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\#\text{ flops} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(1 + \sum_{k=i+1}^{n+1} 2\right)$$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\#\text{ flops} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (2n - 2i + 3)$$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\#\text{ flops} = \sum_{i=1}^{n-1} (2n-2i+3)(n-i)$$

Counting flops for Gaussian elimination:

Input: augmented matrix $A \in \mathbb{R}^{n\times(n+1)}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $m \leftarrow A_{j,i}/A_{i,i}$ $\leftarrow$ 1 flop
    $A(j,i) \leftarrow 0$
    for $k = i+1 : n+1$
      $A(j,k) \leftarrow A(j,k) - m\,A(i,k)$ $\leftarrow$ 2 flops
    end
  end
end

So the number of flops is:

$$\#\text{ flops} = \boxed{\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n}$$

Computing LU decomposition (without pivoting):

Input: $A \in \mathbb{R}^{n\times n}$

$U \leftarrow A$, $L \leftarrow I$ (initialise matrices)
for $j = 1 : n-1$ (loop through pivot columns)
  for $i = j+1 : n$
    $L_{ij} \leftarrow U_{ij}/U_{jj}$ (store multiplier in L matrix)
    for $k = j : n$
      $U_{ik} \leftarrow U_{ik} - L_{ik}U_{jk}$ (update row i of U matrix)
    end for
  end for
end for

Output: Matrices L and U

Computing LU decomposition (without pivoting):

Input: $A \in \mathbb{R}^{n \times n}$
1: $U \leftarrow A$, $L \leftarrow I$ (initialise matrices)
2: **for** $j = 1 : n-1$ (loop through pivot columns)
3: **for** $i = j+1 : n$
4: $L_{ij} \leftarrow U_{ij} / U_{jj}$ (store multiplier in L matrix)
5: **for** $k = j : n$
6: $U_{ik} \leftarrow U_{ik} - L_{ik} U_{jk}$ (update row i of U matrix)
7: **end for**
8: **end for**
9: **end for**
Output: Matrices L and U

- Line 4: one $\boxed{\div}$; Line 6: one $\boxed{+}$, one $\boxed{\times}$

Computing LU decomposition (without pivoting):

$$\#\ \text{flops} = \sum_{j=1}^{n-1} \left[\sum_{i=j+1}^{n} \left(1 + \sum_{k=j}^{n} 2 \right) \right]$$

Computing LU decomposition (without pivoting):

$$\#\ \text{flops} = \sum_{j=1}^{n-1}\left[\sum_{i=j+1}^{n}\left(1+\sum_{k=j}^{n}2\right)\right]$$
$$= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}\left(1+2\sum_{k=1}^{n-j+1}1\right)\right]$$

Computing LU decomposition (without pivoting):

$$\# \text{ flops} = \sum_{j=1}^{n-1} \left[\sum_{i=j+1}^{n} \left(1 + \sum_{k=j}^{n} 2 \right) \right]$$

$$= \sum_{j=1}^{n-1} \left[\sum_{i=1}^{n-j} \left(1 + 2 \sum_{k=1}^{n-j+1} 1 \right) \right]$$

Computing LU decomposition (without pivoting):

$$\begin{aligned}
\#\text{ flops} &= \sum_{j=1}^{n-1}\left[\sum_{i=j+1}^{n}\left(1+\sum_{k=j}^{n}2\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}\left(1+2\sum_{k=1}^{n-j+1}1\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}(1+2(n-j+1))\right]
\end{aligned}$$

Computing LU decomposition (without pivoting):

$$\# \text{ flops} = \sum_{j=1}^{n-1} \left[\sum_{i=j+1}^{n} \left(1 + \sum_{k=j}^{n} 2 \right) \right]$$

$$= \sum_{j=1}^{n-1} \left[\sum_{i=1}^{n-j} \left(1 + 2 \sum_{k=1}^{n-j+1} 1 \right) \right]$$

$$= \sum_{j=1}^{n-1} \left[\sum_{i=1}^{n-j} (1 + 2(n - j + 1)) \right] = \sum_{j=1}^{n-1} \left[(2n - 2j + 3) \sum_{i=1}^{n-j} 1 \right]$$

Computing LU decomposition (without pivoting):

$$
\begin{aligned}
\#\ \text{flops} &= \sum_{j=1}^{n-1}\left[\sum_{i=j+1}^{n}\left(1+\sum_{k=j}^{n}2\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}\left(1+2\sum_{k=1}^{n-j+1}1\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}(1+2(n-j+1))\right] = \sum_{j=1}^{n-1}\left[(2n-2j+3)\sum_{i=1}^{n-j}1\right] \\
&= \sum_{j=1}^{n-1}\left[(2n-2j+3)(n-j)\right]
\end{aligned}
$$

Computing LU decomposition (without pivoting):

$$
\begin{aligned}
\#\text{ flops} &= \sum_{j=1}^{n-1}\left[\sum_{i=j+1}^{n}\left(1+\sum_{k=j}^{n}2\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}\left(1+2\sum_{k=1}^{n-j+1}1\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}(1+2(n-j+1))\right] = \sum_{j=1}^{n-1}\left[(2n-2j+3)\sum_{i=1}^{n-j}1\right] \\
&= \sum_{j=1}^{n-1}[(2n-2j+3)(n-j)] \\
&= 2\sum_{j=1}^{n-1}j^2-(4n+3)\sum_{j=1}^{n-1}j+(2n^2+3n)\sum_{j=1}^{n-1}1
\end{aligned}
$$

Computing LU decomposition (without pivoting):

$$
\begin{aligned}
\text{\# flops} &= \sum_{j=1}^{n-1}\left[\sum_{i=j+1}^{n}\left(1+\sum_{k=j}^{n}2\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}\left(1+2\sum_{k=1}^{n-j+1}1\right)\right] \\
&= \sum_{j=1}^{n-1}\left[\sum_{i=1}^{n-j}(1+2(n-j+1))\right] = \sum_{j=1}^{n-1}\left[(2n-2j+3)\sum_{i=1}^{n-j}1\right] \\
&= \sum_{j=1}^{n-1}[(2n-2j+3)(n-j)] \\
&= 2\sum_{j=1}^{n-1}j^2-(4n+3)\sum_{j=1}^{n-1}j+(2n^2+3n)\sum_{j=1}^{n-1}1 = \boxed{\frac{2}{3}n^3+\frac{1}{2}n^2-\frac{7}{6}n}
\end{aligned}
$$

So the complexity of Gaussian elimination and LU-decomposition are the same.
In either case, to obtain the solution of the linear system, we also need forward / backward substitution:

$$A\mathbf{x} = \mathbf{b} \quad \rightarrow \quad \text{Gaussian elimination} \quad \rightarrow \quad (LA)\mathbf{x} = (L\mathbf{b})$$

with (LA) upper triangular

$$A\mathbf{x} = \mathbf{b} \quad \rightarrow \quad \text{LU-decomposition} \quad \rightarrow \quad (LU)\mathbf{x} = \mathbf{b} \quad \rightarrow$$

first solve $L\mathbf{z} = \mathbf{b}$ (lower triangular)

then solve $U\mathbf{x} = \mathbf{z}$ (upper triangular)

So the complexity of solving the linear system is that of Gaussian elimination / LU-decomposition plus that of forward / backward substitution.

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

Input: $U \in \mathbb{R}^{n\times n}$, $\mathbf{c} \in \mathbb{R}^{n\times 1}$

1: **for** $k = n$ **to** 1 **step** -1 (loop from last row)

2: $x_k \leftarrow c_k$ (initialise vector)

3: **for** $\ell = k+1:n$ (loop through rows beneath k)

4: $x_k \leftarrow x_k - U_{k\ell}x_\ell$ (use x_ℓ already computed)

5: **end for**

6: $x_k \leftarrow x_k / U_{kk}$ (divide by diagonal element)

7: **end for**

Output: Vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$ such that $U\mathbf{x} = \mathbf{c}$ (i.e., $\mathbf{x} = U^{-1}\mathbf{c}$)

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

Input: $U \in \mathbb{R}^{n\times n}$, $\mathbf{c} \in \mathbb{R}^{n\times 1}$
1: **for** $k = n$ **to** 1 **step** -1 (loop from last row)
2: $x_k \leftarrow c_k$ (initialise vector)
3: **for** $\ell = k+1:n$ (loop through rows beneath k)
4: $x_k \leftarrow x_k - U_{k\ell}x_\ell$ (use x_ℓ already computed)
5: **end for**
6: $x_k \leftarrow x_k / U_{kk}$ (divide by diagonal element)
7: **end for**
Output: Vector $\mathbf{x} \in \mathbb{R}^{n\times 1}$ such that $U\mathbf{x} = \mathbf{c}$ (i.e., $\mathbf{x} = U^{-1}\mathbf{c}$)

Algorithm concisely summarised by single formula:

$$x_k = \frac{1}{U_{kk}}\left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell}x_\ell\right) \qquad (k = 1:n)$$

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

$\Rightarrow$ to compute x_k requires $2n - 2k + 1$ flops

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

$\Rightarrow$ to compute x_k requires $2n - 2k + 1$ flops

- Computing all x_k for $k = 1 : n$ requires

$$\text{Cost} = \sum_{k=1}^{n} (2n - 2k + 1)$$

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

$\Rightarrow$ to compute x_k requires $2n - 2k + 1$ flops

- Computing all x_k for $k = 1 : n$ requires

$$\begin{aligned} \text{Cost} &= \sum_{k=1}^{n} (2n - 2k + 1) \\ &= (2n + 1) \sum_{k=1}^{n} 1 - 2 \sum_{k=1}^{n} k \end{aligned}$$

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

$\Rightarrow$ to compute x_k requires $2n - 2k + 1$ flops

- Computing all x_k for $k = 1 : n$ requires

$$\begin{aligned}
\text{Cost} &= \sum_{k=1}^{n} (2n - 2k + 1) \\
&= (2n + 1) \sum_{k=1}^{n} 1 - 2 \sum_{k=1}^{n} k \\
&= (2n + 1)(n) - 2 \left(\frac{n(n+1)}{2} \right)
\end{aligned}$$

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

$\Rightarrow$ to compute x_k requires $2n - 2k + 1$ flops

- Computing all x_k for $k = 1 : n$ requires

$$\begin{aligned} \text{Cost} &= \sum_{k=1}^{n} (2n - 2k + 1) \\ &= (2n+1) \sum_{k=1}^{n} 1 - 2 \sum_{k=1}^{n} k \\ &= (2n+1)(n) - 2 \left(\frac{n(n+1)}{2} \right) \end{aligned}$$

Computing solution of $U\mathbf{x} = \mathbf{c}$ by back substitution:

$$x_k = \frac{1}{U_{kk}} \left(c_k - \sum_{\ell=k+1}^{n} U_{k\ell} x_\ell \right) \qquad (k = n : (-1) : 1)$$

- Computing x_k: $n - k$ $\boxed{\times}$, $n - k - 1$ $\boxed{+}$, 1 $\boxed{-}$, and 1 $\boxed{\div}$,

$\Rightarrow$ to compute x_k requires $2n - 2k + 1$ flops

- Computing all x_k for $k = 1 : n$ requires

$$\begin{aligned} \text{Cost} &= \sum_{k=1}^{n} (2n - 2k + 1) \\ &= (2n + 1) \sum_{k=1}^{n} 1 - 2 \sum_{k=1}^{n} k \\ &= (2n + 1)(n) - 2 \left(\frac{n(n+1)}{2} \right) = \boxed{n^2 \text{ flops}} \end{aligned}$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$f_1(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) &= x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
&= (1) \times x \times x \times x \times x \times x \times x \times x +
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) =& x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
=& (1) \times x \times x \times x \times x \times x \times x \times x + \\
& (-7) \times x \times x \times x \times x \times x \times x +
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) =& x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
=& (1) \times x \times x \times x \times x \times x \times x \times x + \\
& (-7) \times x \times x \times x \times x \times x \times x + \\
& (21) \times x \times x \times x \times x \times x +
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) =& x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
=& (1) \times x \times x \times x \times x \times x \times x \times x + \\
& (-7) \times x \times x \times x \times x \times x \times x + \\
& (21) \times x \times x \times x \times x \times x + \\
& (-35) \times x \times x \times x \times x +
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) = & x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
= & (1) \times x \times x \times x \times x \times x \times x \times x + \\
& (-7) \times x \times x \times x \times x \times x \times x + \\
& (21) \times x \times x \times x \times x \times x + \\
& (-35) \times x \times x \times x \times x + \\
& (35) \times x \times x \times x +
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) =& x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
=& (1) \times x \times x \times x \times x \times x \times x \times x + \\
& (-7) \times x \times x \times x \times x \times x \times x + \\
& (21) \times x \times x \times x \times x \times x + \\
& (-35) \times x \times x \times x \times x + \\
& (35) \times x \times x \times x + \\
& (-21) \times x \times x +
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$\begin{aligned} f_1(x) =& x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\ =& (1) \times x \times x \times x \times x \times x \times x \times x + \\ & (-7) \times x \times x \times x \times x \times x \times x + \\ & (21) \times x \times x \times x \times x \times x + \\ & (-35) \times x \times x \times x \times x + \\ & (35) \times x \times x \times x + \\ & (-21) \times x \times x + \\ & (7) \times x + \end{aligned}$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$
\begin{aligned}
f_1(x) =& x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\
=& (1) \times x \times x \times x \times x \times x \times x \times x + \\
& (-7) \times x \times x \times x \times x \times x \times x + \\
& (21) \times x \times x \times x \times x \times x + \\
& (-35) \times x \times x \times x \times x + \\
& (35) \times x \times x \times x + \\
& (-21) \times x \times x + \\
& (7) \times x + \\
& (-1)
\end{aligned}
$$

Naive polynomial evaluation:

- Consider the polynomial $f_1(x)$ defined by

$$\begin{aligned} f_1(x) = & x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 \\ = & (1) \times x \times x \times x \times x \times x \times x \times x + \\ & (-7) \times x \times x \times x \times x \times x \times x + \\ & (21) \times x \times x \times x \times x \times x + \\ & (-35) \times x \times x \times x \times x + \\ & (35) \times x \times x \times x + \\ & (-21) \times x \times x + \\ & (7) \times x + \\ & (-1) \end{aligned}$$

- 28 $\boxed{\times}$ & 7 $\boxed{+}$ to compute $f_1(x)$

Naive polynomial evaluation vs. nested evaluation:

- $x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$ in nested form is

$$f_2(x) = -1 + x(7 + x(-21 + x(35 + x(-35 + x(21 + x(-7 + x))))))$$

Naive polynomial evaluation vs. nested evaluation:

- $x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$ in nested form is

$$f_2(x) = -1+x(7+x(-21+x(35+x(-35+x(21+x(-7+x))))))$$

- 6 $\boxed{\times}$ & 7 $\boxed{+}$ in $f_2(x)$

Naive polynomial evaluation vs. nested evaluation:

- $x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$ in nested form is

$$f_2(x) = -1+x(7+x(-21+x(35+x(-35+x(21+x(-7+x))))))$$

- 6 $\boxed{\times}$ & 7 $\boxed{+}$ in $f_2(x)$
- $f_1(x) \equiv f_2(x)$ algebraically
- However, $f_2(x)$ has dramatically lower operation count

Naive algorithm for polynomial evaluation:

$$p(x) = a_0 + a_1 x^1 + \cdots + a_n x^n$$

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$

$y \leftarrow a_0$
for $k = 1 : n$
    term $\leftarrow a_k$
    for $\ell = 1 : k$
        term$\leftarrow$ term $\times\, x$
    end for
    $y \leftarrow y +$ term
end for

Output: $y = p(x) = \sum_{k=0}^{n} a_k x^k$

Naive algorithm for polynomial evaluation:

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$

1: $y \leftarrow a_0$
2: **for** $k = 1 : n$
3: term $\leftarrow a_k$
4: **for** $\ell = 1 : k$
5: term$\leftarrow$ term $\times$ x
6: **end for**
7: $y \leftarrow y +$ term
8: **end for**

Output: $y = p(x) = \sum_{k=0}^{n} a_k x^k$

$$p(x) = a_0 + a_1 x^1 + \cdots + a_n x^n$$

- Line 5: one $\boxed{\times}$; Line 7: one $\boxed{+}$

Naive algorithm for polynomial evaluation:

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$

```
1: y ← a_0
2: for k = 1 : n
3:     term ← a_k
4:     for ℓ = 1 : k
5:         term ← term × x
6:     end for
7:     y ← y + term
8: end for
```

Output: $y = p(x) = \sum_{k=0}^{n} a_k x^k$

$$p(x) = a_0 + a_1 x^1 + \cdots + a_n x^n$$

- Line 5: one $\boxed{\times}$; Line 7: one $\boxed{+}$
- Line 5 repeats for $\ell = 1:k$, $k = 1:n$

  $\Rightarrow$ cost is $\sum_{k=1}^{n} \sum_{\ell=1}^{k} 1 = \frac{n(n+1)}{2}$

Naive algorithm for polynomial evaluation:

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$

1: $y \leftarrow a_0$
2: **for** $k = 1:n$
3: term $\leftarrow a_k$
4: **for** $\ell = 1:k$
5: term$\leftarrow$ term $\times\, x$
6: **end for**
7: $y \leftarrow y +$ term
8: **end for**

Output: $y = p(x) = \sum_{k=0}^{n} a_k x^k$

$$p(x) = a_0 + a_1 x^1 + \cdots + a_n x^n$$

- Line 5: one $\boxed{\times}$; Line 7: one $\boxed{+}$
- Line 5 repeats for $\ell = 1:k$, $k = 1:n$
  $$\Rightarrow \text{ cost is } \sum_{k=1}^{n}\sum_{\ell=1}^{k} 1 = \frac{n(n+1)}{2}$$
- Line 7 repeats for $k = 1:n$
  $$\Rightarrow \text{ cost is } \sum_{k=1}^{n} 1 = n \text{ flops}$$

Naive algorithm for polynomial evaluation:

Input: $\{a_k\}_{k=0}^n$, $x \in \mathbb{R}$

1: $y \leftarrow a_0$
2: **for** $k = 1 : n$
3: term $\leftarrow a_k$
4: **for** $\ell = 1 : k$
5: term $\leftarrow$ term $\times\ x$
6: **end for**
7: $y \leftarrow y +$ term
8: **end for**

Output: $y = p(x) = \sum_{k=0}^{n} a_k x^k$

$$p(x) = a_0 + a_1 x^1 + \cdots + a_n x^n$$

- Line 5: one $\boxed{\times}$; Line 7: one $\boxed{+}$
- Line 5 repeats for $\ell = 1 : k$, $k = 1 : n$

  $$\Rightarrow \text{ cost is } \sum_{k=1}^{n} \sum_{\ell=1}^{k} 1 = \frac{n(n+1)}{2}$$

- Line 7 repeats for $k = 1 : n$

  $$\Rightarrow \text{ cost is } \sum_{k=1}^{n} 1 = n \text{ flops}$$

- Total cost is $\boxed{\dfrac{n^2 + 3n}{2} \text{ flops}}$

Horner algorithm for nested polynomial evaluation:

$$\begin{aligned} p(x) &= \sum_{k=0}^{n} a_k x^k \\ &= a_0 + a_1 x^1 + \cdots + a_n x^n \\ &\equiv \underbrace{a_0 + (a_1 + (a_2 + \cdots (a_{n-1} + a_n x) x \cdots) x) x}_{p(x) \text{ rewritten in nested form}} \end{aligned}$$

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$
1: $y \leftarrow a_n$
2: **for** $k = (n-1)$ **to** 0 **step** -1
3: $\quad y \leftarrow x \times y + a_k$
4: **end for**
Output: $y = p(x)$

Horner algorithm for nested polynomial evaluation:

$$\begin{aligned} p(x) &= \sum_{k=0}^{n} a_k x^k \\ &= a_0 + a_1 x^1 + \cdots + a_n x^n \\ &\equiv \underbrace{a_0 + (a_1 + (a_2 + \cdots (a_{n-1} + a_n x)x \cdots)x)x}_{p(x) \text{ rewritten in nested form}} \end{aligned}$$

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$
1: $y \leftarrow a_n$
2: **for** $k = (n-1)$ **to** 0 **step** -1
3: $\quad y \leftarrow x \times y + a_k$
4: **end for**
Output: $y = p(x)$

- Observe loop counter decreasing

Horner algorithm for nested polynomial evaluation:

$$
\begin{aligned}
p(x) &= \sum_{k=0}^{n} a_k x^k \\
&= a_0 + a_1 x^1 + \cdots + a_n x^n \\
&\equiv \underbrace{a_0 + (a_1 + (a_2 + \cdots (a_{n-1} + a_n x)x \cdots)x)x}_{p(x) \text{ rewritten in nested form}}
\end{aligned}
$$

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$
1: $y \leftarrow a_n$
2: **for** $k = (n-1)$ **to** 0 **step** -1
3: $\quad y \leftarrow x \times y + a_k$
4: **end for**
Output: $y = p(x)$

- Observe loop counter decreasing
- Line 3: one $\boxed{+}$ & one $\boxed{\times}$
- Line 3 executes once for $k = 0 : n-1$
- Total cost is $\sum_{k=0}^{n-1} 2$

Horner algorithm for nested polynomial evaluation:

$$
\begin{aligned}
p(x) &= \sum_{k=0}^{n} a_k x^k \\
&= a_0 + a_1 x^1 + \cdots + a_n x^n \\
&\equiv \underbrace{a_0 + (a_1 + (a_2 + \cdots (a_{n-1} + a_n x)x \cdots)x)x}_{p(x) \text{ rewritten in nested form}}
\end{aligned}
$$

Input: $\{a_k\}_{k=0}^{n}$, $x \in \mathbb{R}$
1: $y \leftarrow a_n$
2: **for** $k = (n-1)$ **to** 0 **step** -1
3: $\quad y \leftarrow x \times y + a_k$
4: **end for**
Output: $y = p(x)$

- Observe loop counter decreasing
- Line 3: one $\boxed{+}$ & one $\boxed{\times}$
- Line 3 executes once for $k = 0 : n-1$
- Total cost is $\sum_{k=0}^{n-1} 2 = \boxed{2n \text{ flops}}$

Some conclusions:

- The total cost of solving a linear system with Gaussian elimination is

$$\text{elimination flops} + \text{backward substitution flops} =$$
$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n + n^2 = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

Some conclusions:

- The total cost of solving a linear system with Gaussian elimination is

$$\text{elimination flops} + \text{backward substitution flops} =$$
$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n + n^2 = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

- And with LU-decomposition:

$$\text{decomposition flops} + \text{backward flops} + \text{forward flops} =$$
$$\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{7}{6}n$$

Some conclusions:

- The total cost of solving a linear system with Gaussian elimination is

$$\text{elimination flops} + \text{backward substitution flops} =$$
$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n + n^2 = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

- And with LU-decomposition:

$$\text{decomposition flops} + \text{backward flops} + \text{forward flops} =$$
$$\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{7}{6}n$$

- The cost of evaluating a polynomial of order n is $2n$ – when done in the right way.

Some conclusions:

- The total cost of solving a linear system with Gaussian elimination is

$$\text{elimination flops} + \text{backward substitution flops} =$$

$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n + n^2 = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

- And with LU-decomposition:

$$\text{decomposition flops} + \text{backward flops} + \text{forward flops} =$$

$$\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{7}{6}n$$

- The cost of evaluating a polynomial of order n is $2n$ – when done in the right way.
- A simple re-ordering can reduce the complexity drastically!

Some conclusions:

- The total cost of solving a linear system with Gaussian elimination is

$$\text{elimination flops} + \text{backward substitution flops} =$$
$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n + n^2 = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

- And with LU-decomposition:

$$\text{decomposition flops} + \text{backward flops} + \text{forward flops} =$$
$$\frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{7}{6}n$$

- The cost of evaluating a polynomial of order n is $2n$ – when done in the right way.
- A simple re-ordering can reduce the complexity drastically!

If n is large only the *leading term* matters. . .

The “Big-Oh” symbol is useful to make this observation formal.

Definition ("Big-Oh")

Let $\{x^{(n)}\}$ and $\{y^{(n)}\}$ be two sequences. Then $x^{(n)} = O(y^{(n)})$ (pronounced $x^{(n)}$ is "big-Oh" of $y^{(n)}$) iff there exist constants C and N such that $|x^{(n)}| \leq C|y^{(n)}|$ whenever $n \geq N$.

- $x^{(n)} = O(y^{(n)})$ means $\{x^{(n)}\}$ asymptotically dominated by $\{y^{(n)}\}$
- If $x^{(n)} = O(y^{(n)})$ then $\lim_{n\to\infty} |\frac{x^{(n)}}{y^{(n)}}| \leq C$ for some finite $C \geq 0$
- "Infinite asymptotics": behaviour of sequences as $n \to \infty$
- LU-decomposition is $O(n^3)$.
- Forward / backward substitution is $O(n^2)$.
- "Smart" polynomial evaluation is $O(n)$.

How to visualize this?

If the number of flops f grows as $O(n^p)$, then for large n

$$f \approx \alpha n^p \text{ for some positive } alpha$$

so that

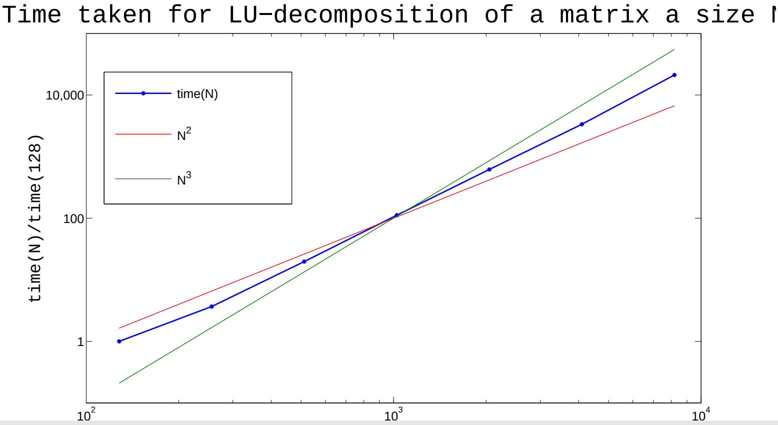
$$\ln(f) \approx \ln(\alpha) + p \ln(n)$$

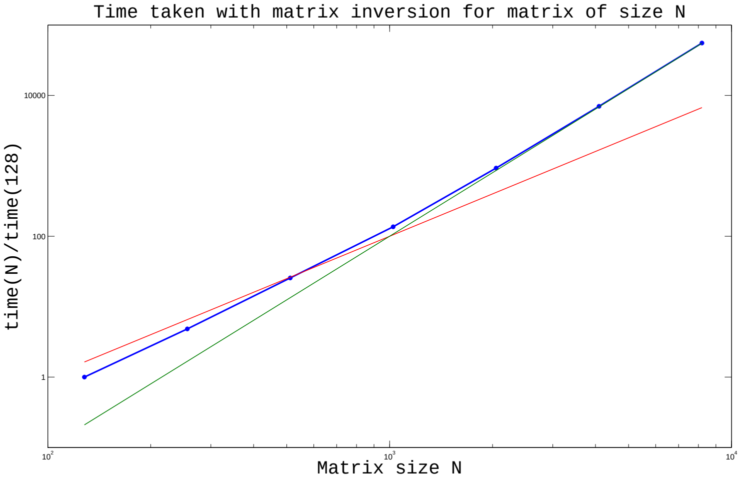
so that $\ln(f)$ depends *linearly* on $\ln(n)$. Plot on a logarithmic scale and find the slope…

Example: LU-decomposition.

$$f = \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n \approx \frac{2}{3}n^3$$

so we expect a straight line with slope 3.
In practice, the slope is in between 2 and 3. Highly optimzed linear algebra routines can achieve a scaling close to $p = 2.4$.




Time taken with matrix inversion for matrix of size N
time(N)/time(128)
10000
100
1
Matrix size N