INMA2111 Discrete mathematics II : Algorithms and complexity

Course notes 8

Dounia Mulders, Cyril de Bodt

April 22, 2015

Random Algorithms (continuation and end)

We saw during last lecture an algorithm to compute π, just by throwing some needles. However, this approach converges quite slowly: the accuracy, measured by the standard deviation, is $\sim \frac{1}{\sqrt{n}}$. We would need a lot of needles...

1 Another way to compute π

Let us first notice that π is simply the area of a unit disc. Thus $\frac{\pi}{4}$ is the area under the red curve of figure 1a. Then a new random algorithm to compute π is simply:

- Pick n random points (x, y) uniformly in $[0, 1]^2$;
- Define $k := |\{(x_i, y_i) : x_i^2 + y_i^2 < 1\}|$;
- $\frac{\pi}{4} \approx \frac{k}{n}$

This algorithm is very similar to the one using the needles, instead that this one can easily be implemented on a computer. Therefore the analyze of convergence remains the same: the accuracy evolve like $\sim \frac{1}{\sqrt{n}}$ (slow convergence).

1.1 A bit smarter

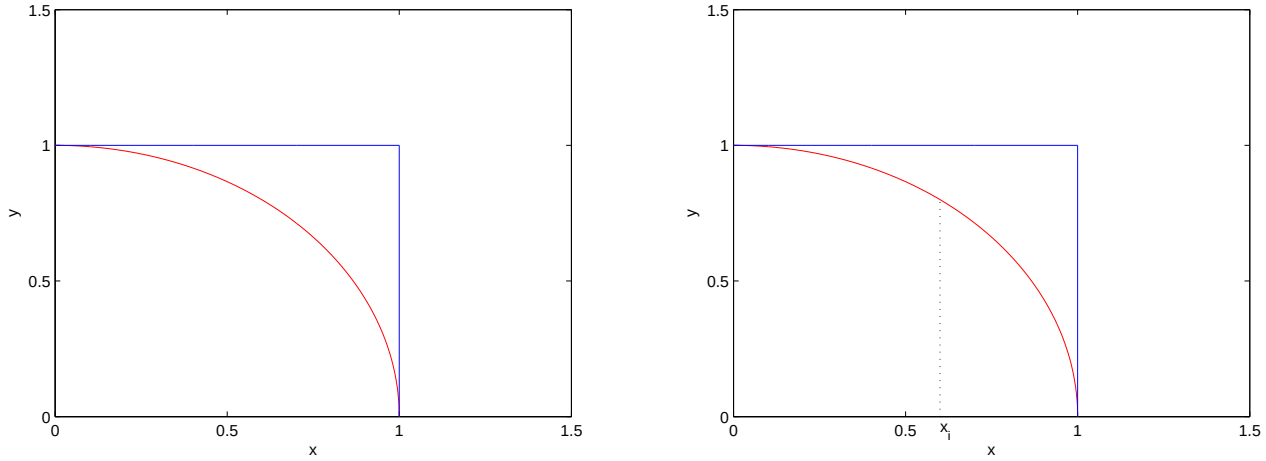
We can easily improve the last naive π estimation:

- Pick x_i randomly between 0 and 1 (see figure 1b);
- Compute[1]

$$\begin{aligned}\frac{\sum_{i=1}^{n}\sqrt{1-x_i^2}}{n} &\approx \mathbb{E}\left[\sqrt{1-x^2}\right] \text{ under uniform distribution of } x \\ &:= \int_0^1 \sqrt{1-x^2}dx \text{ by definition of } \mathbb{E} \\ &= \frac{\pi}{4}\end{aligned}$$

Again the error of this algorithm decreases like $\sim \frac{1}{\sqrt{n}}$.

[1] For a fixed x_i, the probability for the associated y_i to be such that (x_i, y_i) lies under the red curve of figure 1b, is $\frac{\sqrt{1-x_i^2}}{1}$, under uniform distribution. Since we know this probability, we can use it directly instead of picking a random y_i!



(a) $\frac{\pi}{4}$ = area under the red curve. (b)

Figure 1

This method generalizes into a random computation of integrals. 1D integrals are usually better evaluated by deterministic methods (rectangles, simpson, ...). Indeed, the error of the rectangles method (the simplest deterministic method) is in $\frac{1}{n}$, far better than $\frac{1}{\sqrt{n}}$.

2 Why use this random algorithm for integration?

For at least two reasons:

Robin hood effect: For every deterministic method, there are functions for which the method will fail. For example, for high frequency sine (see figure 2a), if the red points are chosen as equally spaced quadrature points, the deterministic method will give a very poor estimate of the integral.

Thus deterministic methods have very bad worst case behaviour. The choice of random points makes all instances "equal".

For kD integrals: $\underbrace{\int\int\dots\int}_{k} f(x_1,\dots,x_k)dx_1\dots dx_k$

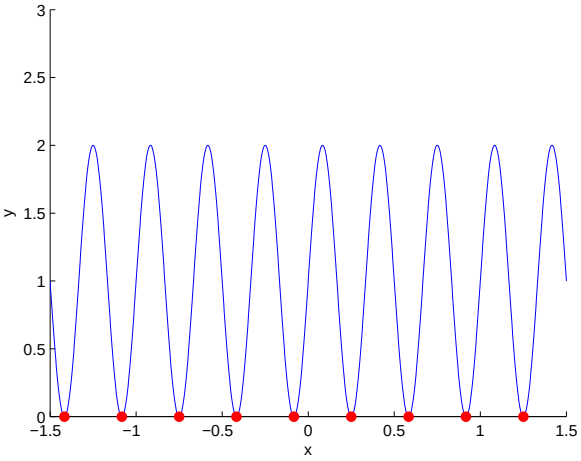
For deterministic methods, n points regularly spaced in $[0,1]^k \rightarrow \sqrt[k]{n}$ points in each dimension (see figure 2b: by choosing the 4 red points, we have 2 blue points in each dimension). Typically, the error will be $\sim \frac{1}{\sqrt[k]{n}}$ (when k increases, we need more points to fill the space[2]).

The random algorithm still behaves in $\frac{1}{\sqrt{n}}$ error (because it depends on the properties of the gaussian distribution) and could possibly be a better choice for triple integrals or more (a hybrid method is even better).

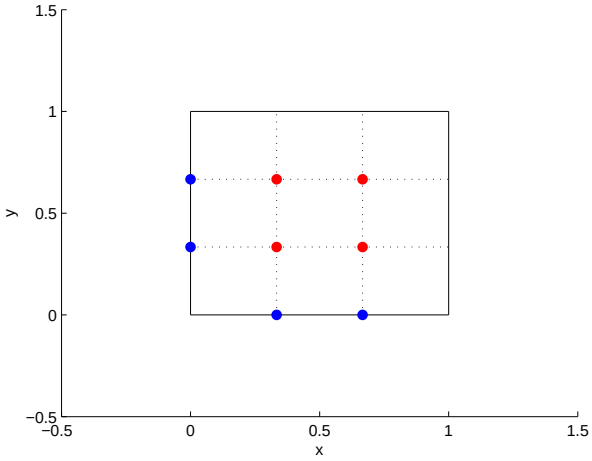
3 An example where randomness is very useful

Suppose you want to check that $A \cdot B = C$, with A, B and C $n \times n$ matrices.

[2]Thus with fixed n, we have less information along each dimension and thus the accuracy decreases.



(a) High frequency sine.



(b)

Figure 2

- Obvious method: compute $A \cdot B$. It costs $\mathcal{O}(n^{2,3,\cdots})$ (using divide and conquer methods). If n is large, then this method might take a lot of time.
- Compute $(A \cdot B)_{i,j}$ ($\mathcal{O}(n)$ time) and compare with $C_{i,j}$ for random i, j. However if there is only one wrong $C_{i,j}$, it is hard to capture it using this method. We would like a method which finds errors with good probability, even if they are really localized.
- If $A \cdot B = C$ then $\forall v \in \mathbb{R}^{n\times 1}$, $\underbrace{A \cdot B \cdot v}_{\mathcal{O}(n^2)} = \underbrace{C \cdot v}_{\mathcal{O}(n^2)}$ (this solution is not localized but spread on the columns, by checking if $A \cdot B = C$ in the direction of v). How to pick v? Randomly in $\{0,1\}^n \rightarrow$ summing a subset of columns of $A \cdot B$ (resp. C) $\Rightarrow A \cdot B \cdot v$ (resp. $C \cdot v$).

  Say there is an error in column j of C (and possibly elsewhere). $\forall v \in \{0,1\}^n$, call $v_{\bar{j}} = v$ except that entry j is flipped ($0 \leftrightarrow 1$).
  - Either $A \cdot B \cdot v \neq C \cdot v$ and $A \cdot B \cdot v_{\bar{j}} \neq C \cdot v_{\bar{j}}$;
  - or $A \cdot B \cdot v = C \cdot v$ and $A \cdot B \cdot v_{\bar{j}} \neq C \cdot v_{\bar{j}}$;
  - or $A \cdot B \cdot v \neq C \cdot v$ and $A \cdot B \cdot v_{\bar{j}} = C \cdot v_{\bar{j}}$;
  - but $A \cdot B \cdot v = C \cdot v$ and $A \cdot B \cdot v_{\bar{j}} = C \cdot v_{\bar{j}}$ is impossible because then

$$\underbrace{A \cdot B \cdot (v - v_{\bar{j}})}_{\pm j^{th} \text{ column of } A\cdot B} = \underbrace{C \cdot (v - v_{\bar{j}})}_{\pm j^{th} \text{ column of } A\cdot B}$$

    which is impossible since there is a mistake in j^{th} column of C!

  Therefore at least half of possible $v \in \{0,1\}^n$ will show an error $\Rightarrow$ Error detected with probability $\geq \frac{1}{2}$ (if there is at least one error).

  This leads us to Freivalds algorithm:

  Freivalds(A, B, C**)** :

  - Pick $v \in \{0,1\}^n$ randomly (using uniform distribution);
  - If $A \cdot B \cdot v = C \cdot v$ then print "$A \cdot B = C$" [maybe];

– If $A \cdot B \cdot v \neq C \cdot v$ then print "$A \cdot B \neq C$" [for sure].

However with this algorithm, false negatives (i.e. $A \cdot B \neq C$ but undected, with probability $\leq \frac{1}{2}$) are possible... To improve Freivalds, we can use amplification of stochastic advantage[3]:

Repeat(A, B, C, k**)** :

– Repeat Freivalds k times;
– If k conclusions "$A \cdot B = C$" then print "$A \cdot B = C$" [maybe];
– If at least one conclusion "$A \cdot B \neq C$" then print "$A \cdot B \neq C$" [for sure].

A mistake will go undetected with probability $\leq \frac{1}{2^k}$. For example, with $k = 10 \Rightarrow \frac{1}{2^k} \approx 10^{-3} = 0.1\%$, with time $\mathcal{O}(kn^2)$ which is better than $\mathcal{O}(n^{2,3,\cdots})$ or $\mathcal{O}(n^3)$.

Amplification of stochastic advantage is very powerful when errors on a decision problem (YES/NO problem[4]) are one-sided: $\exists$ false negative, $\nexists$ false positives, or the contrary (in our case, if we print "$A \cdot B \neq C$", it is for sure).

What now if you get possible false negatives and false positives (for other problems)? Eg

– answer is YES but we conclude YES with probability $\geq \frac{1}{2} + \epsilon$;
– answer is NO but we conclude NO with probability $\geq \frac{1}{2} + \epsilon$;

(the algorithm gives us slightly more often the good answer that when we throw a coin)

Amplification of stochastic advantage still works!

Repeat n times and vote among the answers:

– If $> 50\%$ of YES $\Rightarrow$ output YES;
– if $> 50\%$ of NO $\Rightarrow$ output NO.

What is the probability of error? We can use indicator variables. Let

$$X_i = \begin{cases} 1 & \text{if } i^{th} \text{ run is correct} \\ 0 & \text{if } i^{th} \text{ run is wrong} \end{cases}$$

Then we have $\mathbb{E}[X_i] \geq \frac{1}{2} + \epsilon$. Let us say that $\mathbb{E}[X_i] = \frac{1}{2} + \epsilon$, to study the worst case.
The repeated algorithm is wrong if $\sum_{i=1} X_i < \frac{n}{2}$. But

$$\begin{aligned} \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] &= \left(\frac{1}{2} + \epsilon\right) n \\ \text{Var}\left[X_i\right] &= \mathbb{E}\left[X_i^2\right] - (\mathbb{E}[X_i])^2 \\ &= 1 \cdot \left(\frac{1}{2} + \epsilon\right) + 0 \cdot \left(\frac{1}{2} - \epsilon\right) - \left(\frac{1}{2} + \epsilon\right)^2 \\ &= \frac{1}{4} - \epsilon^2 \\ \text{Var}\left[\sum_{i=1}^{n} X_i\right] &= n \cdot \left(\frac{1}{4} - \epsilon^2\right) \text{ using independence of } X_i\text{'s} \end{aligned}$$

[3] We can try again with other v! We will never be sure that $A \cdot B = C$, but the probability of error will decrease a lot if we test a lot of v.
[4] Answer to the problem is binary; $\neq$ compute π.

Using the Central Limit Theorem, $\sum_{i=1}^{n} X_i \sim$ Gaussian $\left(\left(\frac{1}{2}+\epsilon\right)n, \left(\frac{1}{4}-\epsilon^2\right)n\right)$ if n is large. Using the tables, $\sum_{i=1}^{n} X_i$ has 2.5% of probability to be 2 standard deviation above its mean.

Eg $\epsilon \cdot n = 2 \cdot$ (standard deviation) $= 2\sqrt{n}\sqrt{\frac{1}{4}-\epsilon^2} \Rightarrow n = \frac{4}{\epsilon^2}\left(\frac{1}{4}-\epsilon^2\right) \sim \frac{1}{\epsilon^2}$ for small ϵ. Then we have the wrong answer with probability $\leq 2.5\%$.

If $\epsilon \cdot n = 3 \cdot$(standard deviation) $\Rightarrow n \sim \frac{9}{4\epsilon^2}$, then we get the wrong answer with probability $\leq 0.15\%$.

Thus amplification of stochastic still works, but we need to perform more trials then when only false negatives (or false positive) are appearing. Also we can notice that if $n \sim \frac{1}{\epsilon^2}$ is approximately doubled (so that $n \sim \frac{9}{4\cdot\epsilon^2}$), the probability of error decreases much).

These algorithms provide an uncertain answer (correct with some probability). They are called Monte-Carlo algorithms (they provide possibly wrong answer):

- estimate π;
- estimate kD integrals;
- Freivalds;
- checking that a number is prime (like Freivalds, error is one-sided);
- generating a large prime number[5].

Another sort of algorithms are those that provide a correct answer always, but with random execution time. They are called Las Vegas algorithm. Eg, Random QuickSort, Random selection/Median, ...

Let us see a Las Vegas algorithm for the Eight Queens problem: how to place 8 queens in a chessboard in mutually non-attacking positions, ie

- no two queens in the same row;
- no two queens in the same column;
- no two queens in the same diagonal;

Brute force solution: explore systematically the tree of possibilities, **backtracking** when you reach a dead-end:

- place a queen on the first row;
- place a queen on the second row (but different column);
- place a queen on the third row (but different column);
- $\vdots$
- avoiding same diagonal-positions.

The random 8 queens algorithm makes random choices :

- random column for the 1^{st} queen (in 1^{st} row);
- random column for the 2^{nd} queen (but $\neq$ from the first one and among the valid squares for diagonals)
- ...

[5]The last two items are useful in cryptography.

If there is no more solution to place a new queen, we repeat **from scratch** = no memory of previous attempt (it is the difference with the backtracking solution). We can compute that

$$\mathbb{E}\,(\text{time success}) = \underbrace{\text{Time}_{\text{in case of Success}}}_{\text{time to fill in the chessboard for the final solution}} + \left[\frac{1}{Prob[Success]} - 1\right] \cdot \text{Time}_{\text{in case of failure}}$$

$$\text{because } \frac{1}{Prob[Success]} = \mathbb{E}\,[\text{number of runs to get a success}] = \mathbb{E}\,[\text{number of failure}]$$

Thus we have

$$\mathbb{E}\,(\text{time success}) = 8 + (\frac{1}{0,129...} - 1) \cdot \underbrace{6}_{\text{on average we have 5 or 6 queen placement in a failure}} \text{ queen placements}$$

$$\approx 50 \text{ or } 60 \text{ queen placements}$$

whereas the backtracking solution will get a success after 155 queen placement.

Furthermore the gap grows quickly when 8 queens $\Rightarrow n$ queens (when we increase the number of queens to place). This is at first sight surprising because the random algorithm and the backtracking solution explore the same possibilities but in different orders.

The random algorithm needs less time because placements have nothing "regular".