Random and Parallel Algorithms a journey from Monte Carlo to Las Vegas

Google Maps couldn't help!

Bogdán Zaválnij

Institute of Mathematics and Informatics
University of Pecs

Ljubljana, 2014

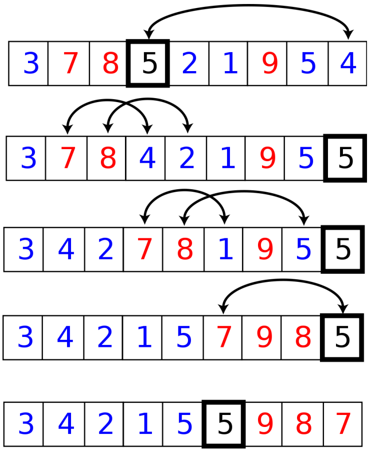
Table of Contents

Table of Contents

Random algorithms you learned earlier: QuickSort

Divide-and-Conquer sorting algorithm

- choose a pivot element
- partition by pivot element (smaller; equal; bigger)
- recursion on the partitions



QuickSort

- Problem:
  - choosing the pivot
  - Is partitioning divides the elements evenly?
  - sometimes we choose the leftmost element
  - $\rightarrow$ poor performance on already sorted array, $O(n^2)$ instead of the expected $O(n \log n)$
- Solution:
  - we choose a random element as the pivot
  - $\rightarrow$ small chance to choose a bad pivot, and so to partition unevenly
  - expected sorting time will be $n \times 2 \log_{4/3} n$, as we will choose the pivot from the middle 50% in half of the cases

Miller–Rabin primality test

For the RSA encryption we need big primes. If we multiply two big prime number, it is a very hard problem, to find the prime factors of this composite number. How do we find big primes? Can we tell a prime from a composite number without knowing its prime factors? Indeed we can! (Maple, acalc: `isprime()` function.)

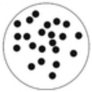
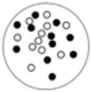
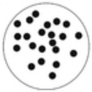
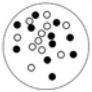
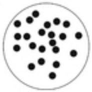
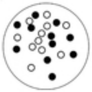
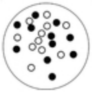
- Let n be prime ($n > 2$). It follows that $n - 1$ is even and we can write it as $n - 1 = 2^s d$
- If we can find an a such that
  - $a^d \not\equiv 1 \pmod{n}$ and
  - $a^{2^r d} \not\equiv -1 \pmod{n}$
  - for all $0 \leq r \leq s - 1$, then n is not prime.
- We call a – a witness for the compositeness of n.
- But how do we find such an a?

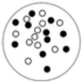
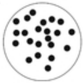
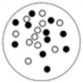
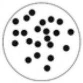
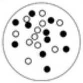
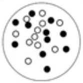
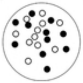
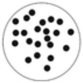
Miller–Rabin primality test

- We choose *a* randomly!
- But what happens, if we choose a “bad” *a*?
- It can be proved, that at least half of the $1 \leq a < n$ are witnesses, if *n* is composite
- So by repeating testing of *n* several times we will tend to choose a “good” witness *a* at least once!
- Repeating the test 50 times the probability of choosing a “bad” *a* is less, than $1/2^{50}$ – it is more probable, that our computer will miscalculate something!

Random choosing – black and white balls

- If we choose a white ball $\rightarrow$ certainly composite.
- If we choose a black ball, perhaps prime!

3	4	5	6	7	8	9
						

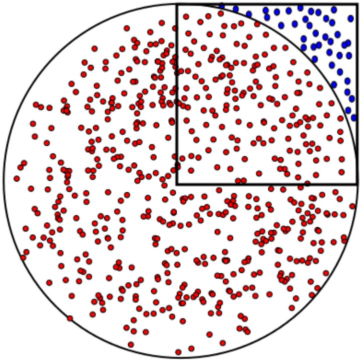
10	11	12	13	14	15	16	17
							

- So far so good!
- But: still no parallel algorithms,
- and no casinos and gambling at all!
- ☹

Table of Contents

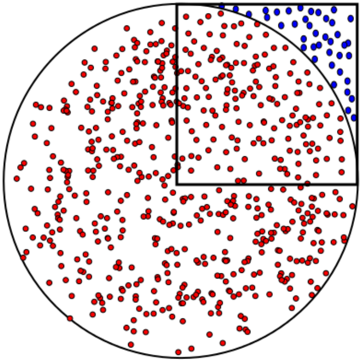
Calculating π on a dartboard

- Take a square frame of size 2×2, and place a circle dartboard $r = 1$ on it
- What is the probability of hitting the dartboard of those hits inside the frame?
  - we consider only the upper-right quarter for simplicity of calculation
  - 140 hits of 180: $140/180 = 0.778$
- The probability is the ratio of the areas:
  - $A_{\text{circle}}/A_{\text{square}} = r^2\pi/s^2 = 1^2\pi/2^2 = \pi/4 = 0.7853981$



Calculating π on a dartboard

- Take a square frame of size 2×2, and place a circle dartboard $r = 1$ on it
- What is the probability of hitting the dartboard of those hits inside the frame?
  - we consider only the upper-right quarter for simplicity of calculation
  - 140 hits of 180: $140/180 = 0.778$
- The probability is the ratio of the areas:
  - $A_{\text{circle}}/A_{\text{square}} = r^2\pi/s^2 = 1^2\pi/2^2 = \pi/4 =$ 0.7853981



Let's calculate π by a computer!

- We generate numbers for x, y coordinates in the upper-right corner
- $\rightarrow$ two random numbers in the range $[0, 1]$
- Is it a hit to the dartboard?
- Calculate the distance to the origin: $\sqrt{x^2 + y^2}$
- If the distance smaller than the radius $r = 1$, then it is a hit!
- (instead of square root calculation we raise both sides to the square and use: $x^2 + y^2 < 1$)

Let's calculate π by a computer! – The C++ code

```
int main(int argc, char **argv){
 double x,y, Pi, error;
 const long long iternum=1000;

 srand48((unsigned)time(0));

 long long sum=0;
 for(long long i=0;i<iternum;++i){
   //x=(double)rand()/RAND_MAX;
   //y=(double)rand()/RAND_MAX;
   x=drand48();
   y=drand48();
   if(x*x+y*y<1) ++sum;
 }
```

The code – cont.

```
  Pi=(4.0*sum)/(iternum);
  error = fabs( Pi-M_PI );

  cout.precision(12);
  cout<<"Pi: \t\t"<<M_PI<<endl;
  cout<<"Pi by MC: \t"<<Pi<<endl;
  cout<<"Error: \t\t"<<fixed<<error<<endl;

  return 0;
}
```

Massage Passing Interface

- Easy extension to the sequential code
- De facto standard in High Performance Computing (supercomputers)
- A dozen function calls to enable MPI (all start with `MPI_`):
  - in the beginning: Initialization
  - sending and receiving data (P2P, Broadcast, Reduction, etc.)
  - at the end: Finalizing
- Predefined data types for portability

- We start *n* separate processes and let them communicate through message passing
- Because we use *n* independent calculations we need to do *n* times less iterations!

The parallel MPI code

```
int main(int argc, char **argv){
  int id, nproc;
  MPI_Status status;
  double x,y, Pi, error;
  long long allsum;
  const long long iternum=1000;

  // Initialize MPI:
  MPI_Init(&argc, &argv);
  // Get my rank:
  MPI_Comm_rank(MPI_COMM_WORLD, &id);
  // Get the total number of processors:
  MPI_Comm_size(MPI_COMM_WORLD, &nproc);
```

The parallel MPI code – cont.

```
srand48((unsigned)time(0));

long long sum=0;
for(long long i=0;i<iternum/nproc;++i){
  x=drand48();
  y=drand48();
  if(x*x+y*y<1) ++sum;
}

//sum the local sum-s into the Master's allsum
MPI_Reduce(&sum, &allsum, 1, MPI_LONG_LONG, MPI_SUM,
0, MPI_COMM_WORLD);
```

The parallel MPI code – cont.

```
//Master writes out the calculated Pi
if(id==0){
  //calculate Pi, compare to the Pi in math.h
  Pi=(4.0*allsum)/(iternum);
  error = fabs( Pi-M_PI );
  cout.precision(12);
  cout<<"Pi: \t\t"<<M_PI<<endl;
  cout<<"Pi by MC: \t"<<Pi<<endl;
  cout<<"Error: \t\t"<<fixed<<error<<endl;
}

// Terminate MPI:
MPI_Finalize();
return 0;
}
```

Numerical Integration

Suppose we want to calculate the integral of a smooth function *f* on the interval $[a, b]$ on the real axis:

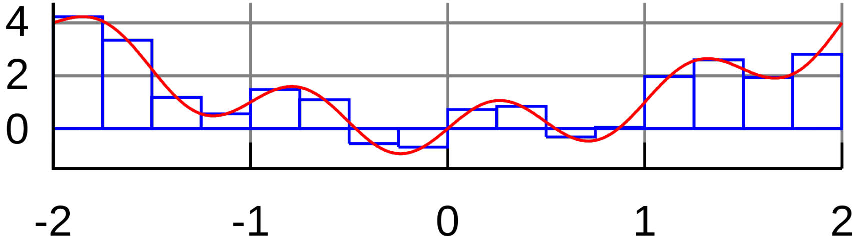
$$I = \int_a^b f(x)\,dx$$

In numerical methods we choose points in the desired interval and interpolate the function using function values at these points. We use N equidistant points starting from a:

$$x_n = a + n\frac{(b-a)}{N}.$$

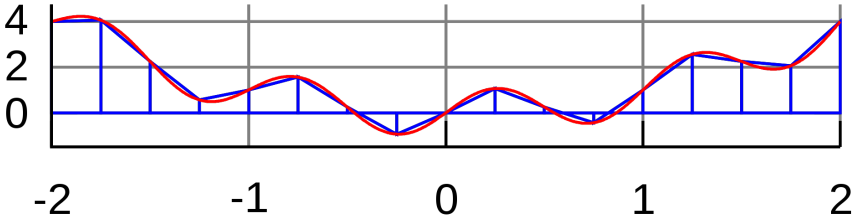
Using the rectangular formula

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{N}\sum_{n=0}^{N-1} f(x_n)$$



Using the trapezoidal formula

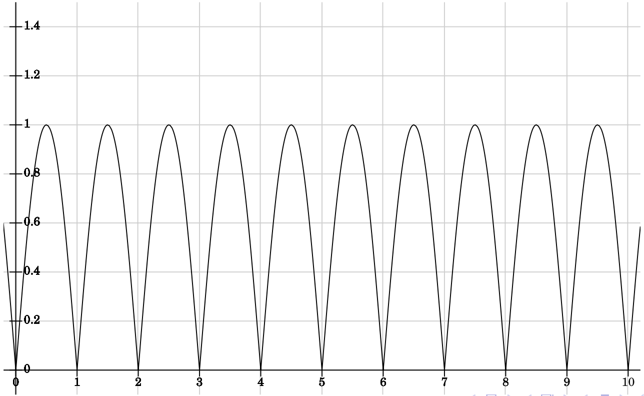
$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{N}\frac{1}{2}\sum_{n=0}^{N-1}(f(x_{n+1})+f(x_n))$$



Problems

Consider the function $y = |(\sin(\pi x)|$, and we would like to integrate it over $[0, 10]$ using 11 sample points: $x_n = 0, 1, 2, \ldots, 10$:

$$I = \int_0^{10} |(\sin(\pi x)| \, dx$$



Monte Carlo Integration

The techniques described called the Quadrature Formulas, where we choose some w_n weights for the integration:

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{N}\sum_{n=0}^{N-1} w_n f(x_n)$$

The Monte Carlo Integration is a variation of this method, where we choose all the weights $w_n = 1$, and we choose the x_n points randomly! The benefits are:

- In higher dimension the exact numerical methods need so many points that we cannot calculate with them
- In higher dimension the exact numerical methods tend to be less and less accurate, while the error of the Monte Carlo Integration is independent of dimension: $O(\sqrt{1/N})$
- The Monte Carlo method is easy to implement, and. . .

Monte Carlo Integration

The techniques described called the Quadrature Formulas, where we choose some w_n weights for the integration:

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{N} \sum_{n=0}^{N-1} w_n f(x_n)$$

The Monte Carlo Integration is a variation of this method, where we choose all the weights $w_n = 1$, and we choose the x_n points randomly! The benefits are:

- In higher dimension the exact numerical methods need so many points that we cannot calculate with them
- In higher dimension the exact numerical methods tend to be less and less accurate, while the error of the Monte Carlo Integration is independent of dimension: $O(\sqrt{1/N})$
- The Monte Carlo method is easy to implement, and. . .

and. . . it is easy to parallelize!

- Independent processes can calculate with any random points
- Only at the end we need some communication (reduction)
- $\rightarrow$ embarrassingly parallel!

The Monte Carlo method is even more interesting when we would like to calculate over some “interesting” volume.

- In this case we use a “frame” of an “easier” shape which includes the target shape, and
- use the w_n weights indicating whether the given point is inside the shape (than $w_n = 1$), or outside it ($w_n = 0$).

The calculation of π used the integration of $f(x) = 1$ over a circle shape!

and. . . it is easy to parallelize!

- Independent processes can calculate with any random points
- Only at the end we need some communication (reduction)
- $\rightarrow$ embarrassingly parallel!

The Monte Carlo method is even more interesting when we would like to calculate over some “interesting” volume.

- In this case we use a “frame” of an “easier” shape which includes the target shape, and
- use the w_n weights indicating whether the given point is inside the shape (than $w_n = 1$), or outside it ($w_n = 0$).

The calculation of π used the integration of $f(x) = 1$ over a circle shape!

and. . . it is easy to parallelize!

- Independent processes can calculate with any random points
- Only at the end we need some communication (reduction)
- $\rightarrow$ embarrassingly parallel!

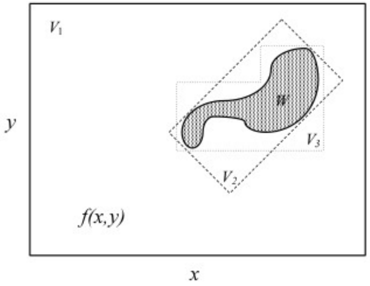
The Monte Carlo method is even more interesting when we would like to calculate over some “interesting” volume.

- In this case we use a “frame” of an “easier” shape which includes the target shape, and
- use the w_n weights indicating whether the given point is inside the shape (than $w_n = 1$), or outside it ($w_n = 0$).

The calculation of π used the integration of $f(x) = 1$ over a circle shape!

Crucial points of the method

Try to minimize the size of the frame, as the opposite will increase the error! The figure shows three possible regions V that might be used to sample a complicated region W. V_1 is a poor choice, V_2 or V_3 better.



The (original) error is:

$$\sigma = \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}},$$

where

$$\langle f \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} f(x_i), \quad \langle f^2 \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} f^2(x_i)$$

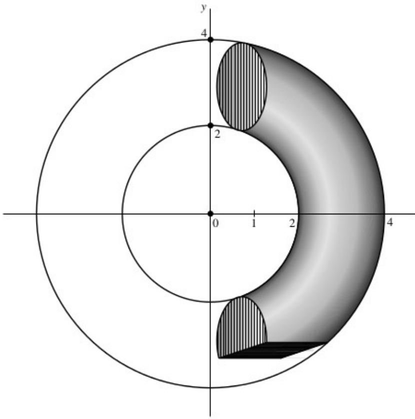
An interesting problem: center of mass

We want to estimate the following integrals over the interior of the complicated object:

$$\int \rho \, dx \, dy \, dz$$

$$\int x\rho \, dx \, dy \, dz \quad \int y\rho \, dx \, dy \, dz \quad \int z\rho \, dx \, dy \, dz$$

The coordinates of the center of mass will be the ratio of the latter three integrals (linear moments) to the first one (the weight).



An interesting problem – cont.

Let $\rho = 1$ (we could use any function instead!) The inequality for the interior of a torus:

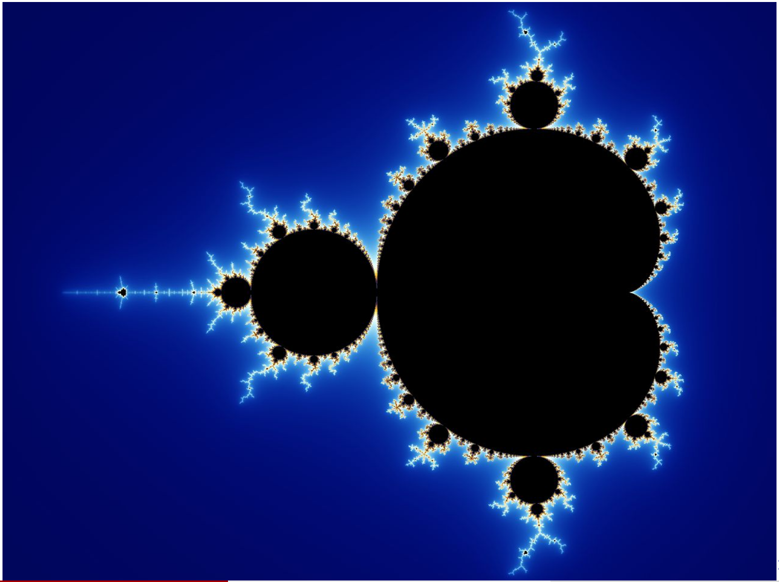
$$\left(R - \sqrt{x^2 + y^2}\right)^2 + z^2 \leq r^2$$

Our torus has $R = 3$ and $r = 1$. The bounding box intersects at $x = 1$ and $y = -3$. So we have three inequalities:

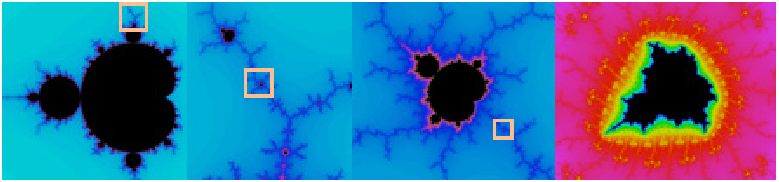
$$\left(3 - \sqrt{x^2 + y^2}\right)^2 + z^2 \leq 1, \quad x \geq 1, \quad y \geq -3$$

Actually if we choose V, enclosing the piece-of-torus W, as the rectangular box extending from 1 to 4 in x, -3 to 4 in y and -1 to 1 in z, then we need only the first inequality. Thus we will choose random points from this V region, and complete the Monte Carlo Integration.

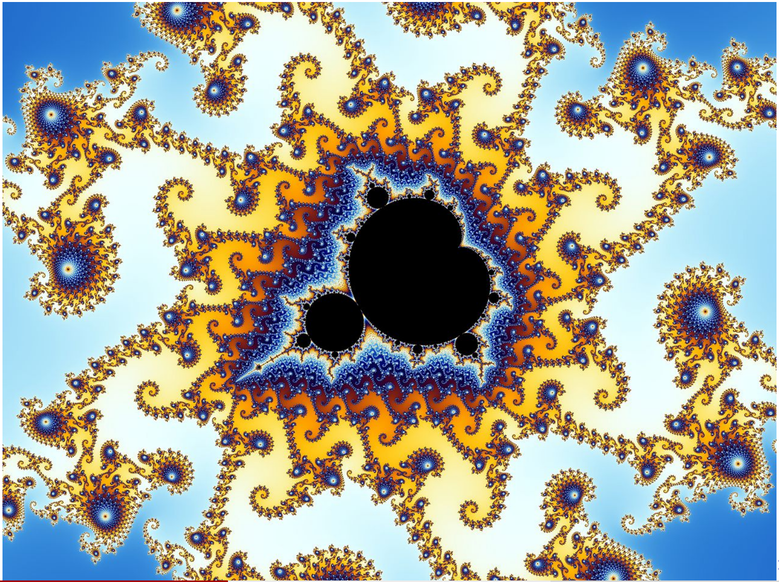
The Mandelbrot set



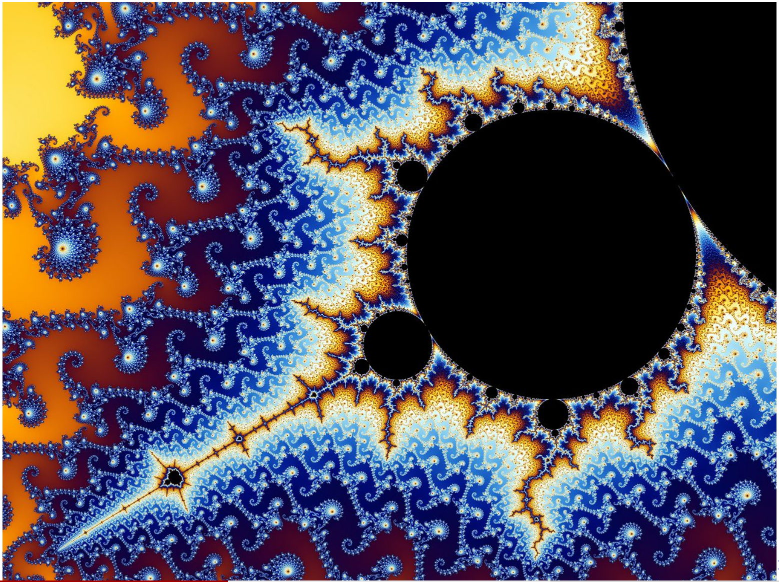
The Mandelbrot set



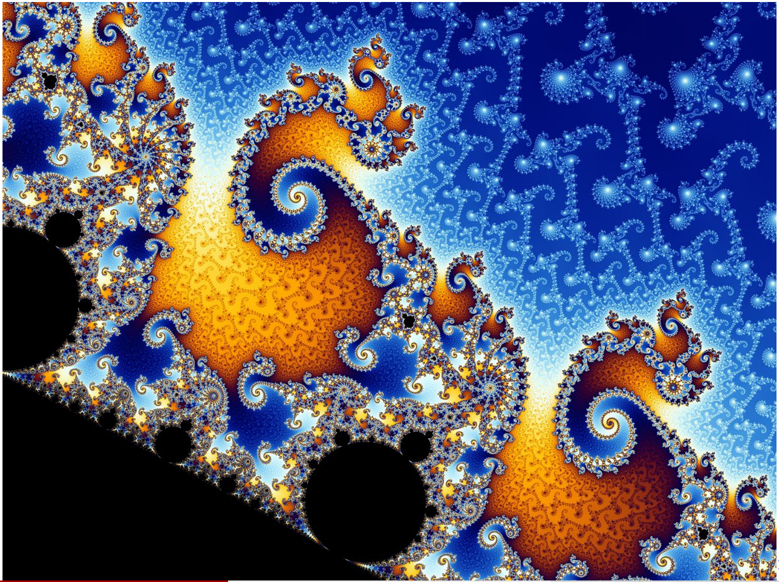
The Mandelbrot set – fractal shape



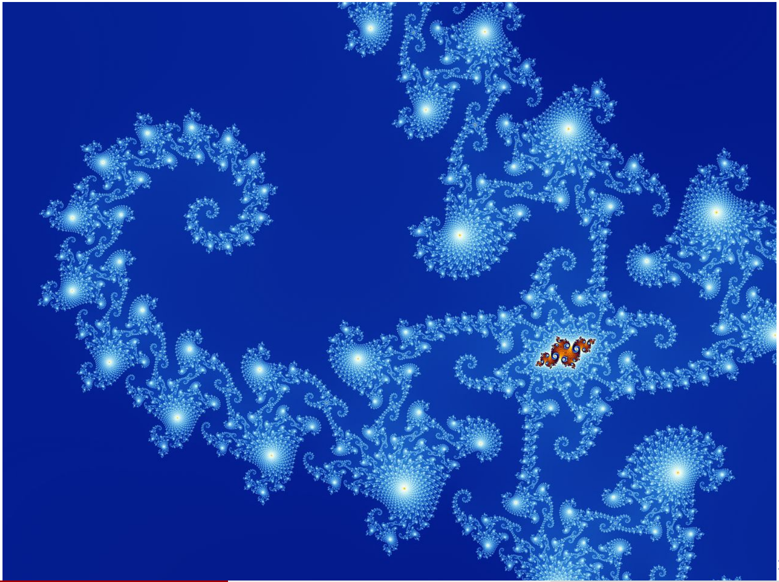
The Mandelbrot set



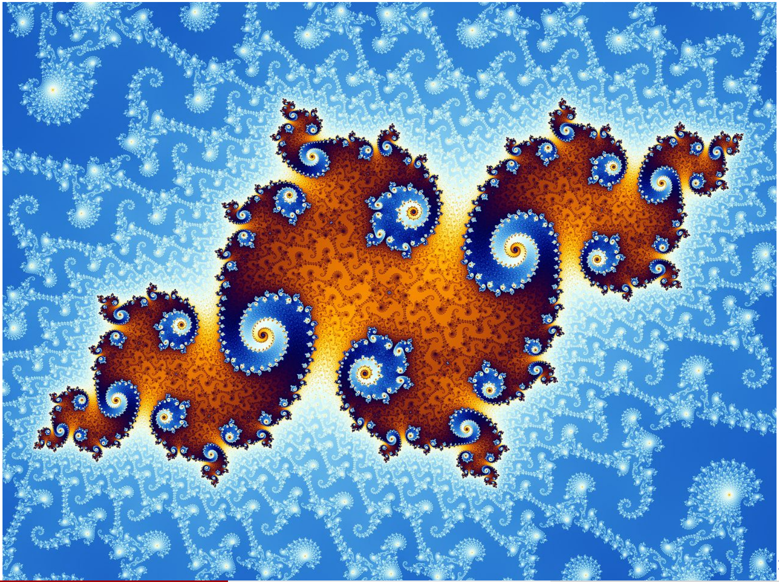
The Mandelbrot set



The Mandelbrot set



The Mandelbrot set



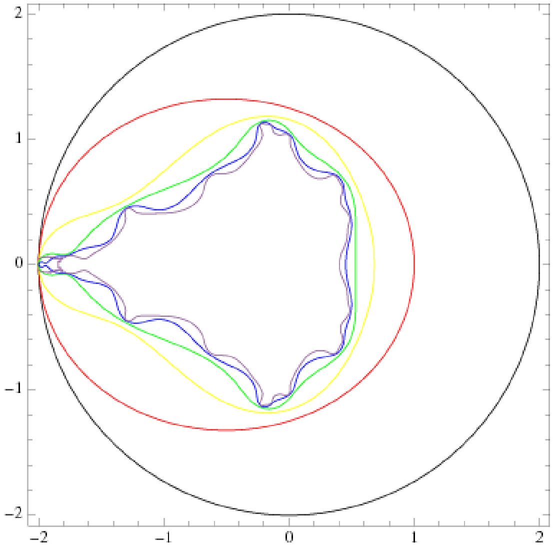
The Mandelbrot set

The Mandelbrot set is the set of values of c in the complex plane for which the orbit of 0 under iteration of the complex quadratic polynomial

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

remains bounded. That is, a complex number c is part of the Mandelbrot set if, when starting with $z_0 = 0$ and applying the iteration repeatedly, the absolute value of z_n remains bounded however large n gets.
For example, letting $c = 1$ gives the sequence $0, 1, 2, 5, 26, \ldots,$ which tends to infinity. As this sequence is unbounded, 1 is not an element of the Mandelbrot set. On the other hand, $c = -1$ gives the sequence $0, -1, 0, -1, 0. \ldots,$ which is bounded, and so -1 belongs to the Mandelbrot set.
(Coloring is artificial, usually denotes the number of steps.)

The Mandelbrot set



The Mandelbrot set

The Mandelbrot set M is defined by a family of complex quadratic polynomials

$$P_c : \mathbb{C} \to \mathbb{C} \text{ given by } P_c : z \mapsto z^2 + c,$$

where c is a complex parameter. $P_c^n(z)$ denotes the nth iterate of $P_c(z)$ The Mandelbrot set is a compact set, contained in the closed disk of radius 2 around the origin. In fact, a point c belongs to the Mandelbrot set if and only if

$$|P_c^n(0)| \leq 2 \text{ for all } n \geq 0.$$

In other words, if the absolute value of $P_c^n(0)$ ever becomes larger than 2, the sequence will escape to infinity.

The Mandelbrot set

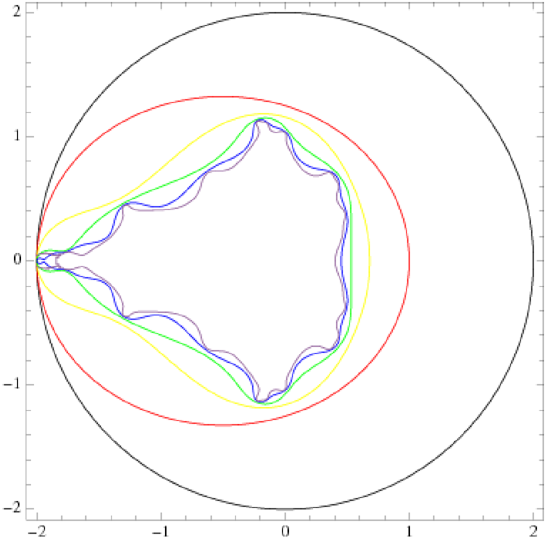
Computer programs iterate given number of steps and inspect if the sequence escapes the disk of radius 2. The number is: $z_x + z_y i$,

```
while( (n<iter_n) && (zx*zx + zy*zy)<4 )
 {
   new_zx = zx*zx - zy*zy + cx;
   zy = 2*zx*zy + cy;
   zx = new_zx;
   n++;
 }
```

We can check if `n==iter_n` for bound or escape, or color by the value of `n`.

Problem: calculate the area of the Mandelbrot set

- How do we calculate the area of a fractal shape?
- By Monte Carlo method!



Problem: calculate the area of the Mandelbrot set

- How do we calculate the area of a fractal shape?
- By Monte Carlo method!

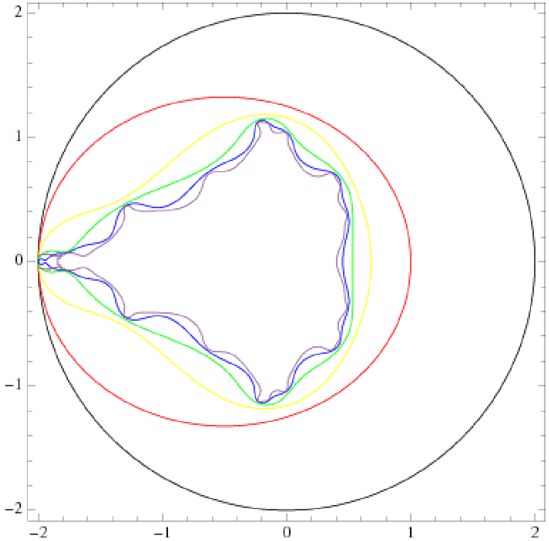


Table of Contents

Randomness in Algorithms

How do we define the “Monte Carlo Method”?

- Literally any algorithm, which use random numbers (or takes random decision)
- Probabilistic Monte Carlo Method – the random numbers simulate directly the physical phenomena we would like to observe (direct simulation by MC)
  - nuclear physics
  - random fluctuations in the telephone traffic
  - flood control and dam construction
  - bottlenecks and queueing systems in industrial production processes
  - study of epidemics
- Deterministic Monte Carlo Method – in problems we can formulate in theoretical language, but cannot solve by theoretical means (MC algorithms)
  - the Monte Carlo integration is an example

Randomness in Algorithms

How do we define the “Monte Carlo Method”?

- Literally any algorithm, which use random numbers (or takes random decision)
- Probabilistic Monte Carlo Method – the random numbers simulate directly the physical phenomena we would like to observe (direct simulation by MC)
  - nuclear physics
  - random fluctuations in the telephone traffic
  - flood control and dam construction
  - bottlenecks and queueing systems in industrial production processes
  - study of epidemics
- Deterministic Monte Carlo Method – in problems we can formulate in theoretical language, but cannot solve by theoretical means (MC algorithms)
  - the Monte Carlo integration is an example

Randomness in Algorithms

How do we define the “Monte Carlo Method”?

- Literally any algorithm, which use random numbers (or takes random decision)
- Probabilistic Monte Carlo Method – the random numbers simulate directly the physical phenomena we would like to observe (direct simulation by MC)
  - nuclear physics
  - random fluctuations in the telephone traffic
  - flood control and dam construction
  - bottlenecks and queueing systems in industrial production processes
  - study of epidemics
- Deterministic Monte Carlo Method – in problems we can formulate in theoretical language, but cannot solve by theoretical means (MC algorithms)
  - the Monte Carlo integration is an example

Randomness in Algorithms

How do we define the “Monte Carlo Method”?

- Literally any algorithm, which use random numbers (or takes random decision)
- Probabilistic Monte Carlo Method – the random numbers simulate directly the physical phenomena we would like to observe (direct simulation by MC)
  - nuclear physics
  - random fluctuations in the telephone traffic
  - flood control and dam construction
  - bottlenecks and queueing systems in industrial production processes
  - study of epidemics
- Deterministic Monte Carlo Method – in problems we can formulate in theoretical language, but cannot solve by theoretical means (MC algorithms)
  - the Monte Carlo integration is an example

Categorization

We can categorize the deterministic Monte Carlo Method by the nature of its error

- Two sided error
  - typical for engineering simulations (the MC integration is an example)
  - we have a $\pm$ error
  - the magnitude of the error controlled by the number of sampling points
  - we can stop at any time
- One sided error
  - the primality tests are good examples
  - we “ask” something, and get a probability answer with one sided error
  - if the answer is “not prime”, it is 100% certain, if the answer is “prime”, it is probable
  - we can speak about one sided error on true side, or false side

Categorization

We can categorize the deterministic Monte Carlo Method by the nature of its error

- Two sided error
  - typical for engineering simulations (the MC integration is an example)
  - we have a $\pm$ error
  - the magnitude of the error controlled by the number of sampling points
  - we can stop at any time
- One sided error
  - the primality tests are good examples
  - we “ask” something, and get a probability answer with one sided error
  - if the answer is “not prime”, it is 100% certain, if the answer is “prime”, it is probable
  - we can speak about one sided error on true side, or false side

Categorization

We can categorize the deterministic Monte Carlo Method by the nature of its error

- Two sided error
  - typical for engineering simulations (the MC integration is an example)
  - we have a $\pm$ error
  - the magnitude of the error controlled by the number of sampling points
  - we can stop at any time
- One sided error
  - the primality tests are good examples
  - we “ask” something, and get a probability answer with one sided error
  - if the answer is “not prime”, it is 100% certain, if the answer is “prime”, it is probable
  - we can speak about one sided error on true side, or false side

Categorization

We can categorize the deterministic Monte Carlo Method by the nature of its error

- Two sided error
    - typical for engineering simulations (the MC integration is an example)
    - we have a $\pm$ error
    - the magnitude of the error controlled by the number of sampling points
    - we can stop at any time
- One sided error
    - the primality tests are good examples
    - we “ask” something, and get a probability answer with one sided error
    - if the answer is “not prime”, it is 100% certain, if the answer is “prime”, it is probable
    - we can speak about one sided error on true side, or false side

$\rightarrow$ easy parallel implementation for both (mostly the first)

Categorization – cont.

- Zero sided error
- $\rightarrow$ the algorithm runs with no error at all
- QuickSort is a good example – we always get a sorted sequence
- Definition: A is a Las Vegas algorithm for a problem class Π, if and only if
    - if for a given problem instance $\pi \in \Pi$, algorithm A terminates returning solution s, s is guaranteed to be a correct solution of π
    - for any given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable
- (we can speak of certainly terminating algorithms as well)

Categorization – cont.

- Zero sided error
- $\rightarrow$ the algorithm runs with no error at all
- QuickSort is a good example – we always get a sorted sequence
- Definition: A is a Las Vegas algorithm for a problem class Π, if and only if
    - if for a given problem instance $\pi \in \Pi$, algorithm A terminates returning solution s, s is guaranteed to be a correct solution of π
    - for any given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable
- (we can speak of certainly terminating algorithms as well)

Categorization – cont.

- Zero sided error
- $\rightarrow$ the algorithm runs with no error at all
- QuickSort is a good example – we always get a sorted sequence
- Definition: A is a Las Vegas algorithm for a problem class Π, if and only if
    - if for a given problem instance $\pi \in \Pi$, algorithm A terminates returning solution s, s is guaranteed to be a correct solution of π
    - for any given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable
- (we can speak of certainly terminating algorithms as well)

Categorization – cont.

- Zero sided error
- $\rightarrow$ the algorithm runs with no error at all
- QuickSort is a good example – we always get a sorted sequence
- Definition: *A* is a Las Vegas algorithm for a problem class Π, if and only if
    - if for a given problem instance $\pi \in \Pi$, algorithm *A* terminates returning solution *s*, *s* is guaranteed to be a correct solution of π
    - for any given instance $\pi \in \Pi$, the run-time of *A* applied to π is a random variable
- (we can speak of certainly terminating algorithms as well)

These algorithms called the Las Vegas algorithms

Table of Contents

Parallel Las Vegas Algorithms

- Parallelization of MC is straightforward
- But how do we make a parallel Las Vegas Algorithm?
- Different approaches, usually start *n* different appliances of the algorithm, with different “starting points” (starting parameters), and let the first “win”!
- Parallel multi-walk Las Vegas algorithm, usually in discrete optimization

Parallel Las Vegas Algorithms – cont.

- Definition: A' is a multi-walk parallel Las Vegas algorithm for a problem class Π, if and only if
  - It consists of n instances of sequential Las Vegas algorithm A for Π, say $A_1, \ldots, A_n$.
  - If, for a given problem instance $\pi \in \Pi$, there exists at least one $i \in [1, n]$ such that A_i terminates let $A_m, m \in [1, n]$ be the instance of A terminating with the minimal runtime and let s be the solution returned by A_m. Then algorithm A' terminates in the same time as A_m and returns solution s.
  - If, for a given problem instance $\pi \in \Pi$, all $A_i, i \in [1.n]$ do not terminate then A' does not terminate.

- Cormen, T., et al. *Introduction to Algorithms.* The MIT Press. 2009.
- Hammersley, J.M. and Handscomb, D.C. *Monte Carlo Methods.* London. 1975. (1964.)
- Press, W., et al. *Numerical Recepies.* Cambridge University Press. 2007.
- Thijssen, J.M. *Computational Physics.* Cambridge University Press. 2003.
- Truchet, C., et al. *Prediction of Parallel Speed-ups for Las Vegas Algorithms.* `http://arxiv.org` 2012.

Thank you for your attention!

Questions?

- Cormen, T., et al. *Introduction to Algorithms.* The MIT Press. 2009.
- Hammersley, J.M. and Handscomb, D.C. *Monte Carlo Methods.* London. 1975. (1964.)
- Press, W., et al. *Numerical Recepies.* Cambridge University Press. 2007.
- Thijssen, J.M. *Computational Physics.* Cambridge University Press. 2003.
- Truchet, C., et al. *Prediction of Parallel Speed-ups for Las Vegas Algorithms.* `http://arxiv.org` 2012.

Thank you for your attention!

Questions?

- Cormen, T., et al. *Introduction to Algorithms.* The MIT Press. 2009.
- Hammersley, J.M. and Handscomb, D.C. *Monte Carlo Methods.* London. 1975. (1964.)
- Press, W., et al. *Numerical Recepies.* Cambridge University Press. 2007.
- Thijssen, J.M. *Computational Physics.* Cambridge University Press. 2003.
- Truchet, C., et al. *Prediction of Parallel Speed-ups for Las Vegas Algorithms.* `http://arxiv.org` 2012.

Thank you for your attention!

Questions?