Μέθοδοι υπολογισμού του π
Benchmark Julia vs Matlab

Νικόλαος Κουκουδάκης

Εαρινό εξάμηνο 2021

Στο βιβλίο των Van Loan - Fan περιγράφονται αναλυτικά οι παρακάτω μέθοδοι.

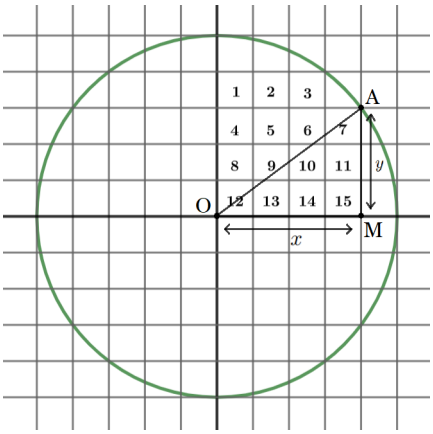
1 Κάλυψη κύκλου με τετραγωνάκια

Θεωρούμε κύκλο της μορφής $x^2 + y^2 = n^2$ με N τετραγωνάκια 1×1.

Έχουμε: $E_{κυκλου} = \pi n^2$ το οποίο προσεγγίζεται με τον αριθμό των τετραγώνων που βρίσκονται ολόκληρα εντός του κύκλου.

Άρα: $N \approx \pi n^2$

Οπότε προσέγγιση του π είναι το $\frac{N}{n^2}$.



Υπολογίζουμε το πλήθος των ολόκληρων τετραγώνων πρώτα στο 1ο τεταρτημόριο.

Στην κ γραμμή ισχύει από Πυθαγόρειο Θεώρημα στο $\overset{\triangle}{OAM}$:

$$x^2 + y^2 = n^2 \iff x = \sqrt{n^2 - y^2}$$

αλλά για να είναι ο x ακέραιος, πρέπει:

$$x = floor(\sqrt{n^2 - \kappa^2})$$

Κάπως έτσι καταλήγουμε στον παρακάτω κώδικα σε Matlab:

```
function err = pi_squares(n)
  N = 0;
  x = 0;
  for k = 1:n
    x = floor(sqrt(n*n - k*k));
    N = N + x;
  end
  N = 4*N;
  myPi = N/(n*n);
  err = abs((myPi-pi)/pi);
```

Και σε Julia:

```
function pi_squares(n)
  N = 0
  x = 0
  for k in 1:n
    x = floor(sqrt(n*n - k*k))
    N = N + x
  end
  N = 4*N
  myPi = N/(n*n)
  err = abs((myPi-pi)/pi)
  return err
end
```

Πολυπλοκότητα: $\mathrm{O}(n)$

Η ακρίβεια του αλγορίθμου (ανεξάρτητη της γλώσσας):

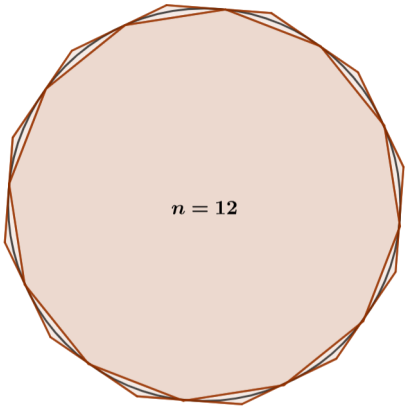
n	Τάξη σφάλματος
1,000,000	10^{-6}
10,000,000	10^{-7}
100,000,000	10^{-8}
1,000,000,000	10^{-9}

Οι επιδόσεις των δύο αλγορίθμων:

n	*Matlab*	*Octave*	*Julia*
1,000,000	$0.04sec$	$15.7sec$	$0.006sec$
10,000,000	$0.12sec$	$2min\ 38sec$	$0.05sec$
100,000,000	$1.1sec$	$31min$	$0.5sec$
1,000,000,000	$10.8sec$	$4h\ 36min$	$4.8sec$

2 Με εμβαδά εγγεγραμμένων και περιγεγραμμένων πολυγώνων

Θεωρούμε κύκλο ακτίνας 1 και γράφουμε το εγγεγραμμένο και περιγεγραμμένο n-γωνο. Έστω $E_{\varepsilon\gamma\gamma} = A_n$ και $E_{\pi\varepsilon\rho\iota\gamma} = B_n$.



Από ιδιότητες γεωμετρίας:

$$A_n = \frac{n}{2} sin\left(\frac{2\pi}{n}\right)$$

$$B_n = ntan\left(\frac{\pi}{n}\right)$$

Οπότε:

$$p_n = \frac{A_n + B_n}{2}$$

Ο κώδικας σε Matlab:

```
function err = pi_polygons(n)
  A = n * sin(2*pi/n) / 2;
  B = n * tan(pi/n);
  myPi = (A + B) / 2;
  err = abs((pi-myPi)/pi);
```

Και σε Julia:

```
function pi_polygons(n)
  A = n * sin(2*pi/n) / 2
  B = n * tan(pi/n)
  myPi = (A + B) / 2
  err = abs((pi-myPi)/pi)
  return err
end
```

Πολυπλοκότητα: O(1)

Η ακρίβεια του αλγορίθμου:

n	Τάξη σφάλματος
100,000	10^{-10}
1,000,000	10^{-12}
10,000,000	10^{-14}
100,000,000	10^{-16}

Οι επιδόσεις των δύο αλγορίθμων:

n	$Matlab$	$Octave$	$Julia$
100,000	$0.001sec$	$0.001sec$	$0.001sec$
1,000,000	$0.001sec$	$0.001sec$	$0.001sec$
10,000,000	$0.001sec$	$0.001sec$	$0.001sec$
100,000,000	$0.001sec$	$0.001sec$	$0.001sec$

3 Ρητή προσέγγιση

Θα προσεγγίσουμε το π ως πηλίκο

$$\pi \approx \frac{p}{q}, \quad p, q \in \mathbb{N}$$

με

$$1 \leq p, q \leq M$$

3.1 Brute force

Ο 1ος τρόπος δοκιμάζει όλους τους πιθανούς συνδυασμούς των p και q και καταλήγει σε αυτόν με το μικρότερο σφάλμα.

Ο κώδικας σε Matlab:

```
function err = pi_q_brute_force(M)
  pbest = 1;
  qbest = 1;
  err = abs((pbest/qbest-pi)/pi);
  for q = 1:M
    p0 = 1;
    e0 = abs((p0/q-pi)/pi);
    for p = 1:M
      if abs((p/q-pi)/pi) < e0
        p0 = p;
        e0 = abs((p/q-pi)/pi);
        if e0 < err
          pbest = p;
          qbest = q;
          err = e0;
        end
      end
    end
  end
  myPi = pbest/qbest;
```

Και σε Julia:

```
function pi_q_brute_force(M)
  pbest = 1;
  qbest = 1;
  err = abs((pbest/qbest-pi)/pi);
  for q in 1:M
    p0 = 1;
    e0 = abs((p0/q-pi)/pi);
    for p in 1:M
      if abs((p/q-pi)/pi) < e0
```

```
10          p0 = p;
11          e0 = abs((p/q-pi)/pi);
12          if e0 < err
13            pbest = p;
14            qbest = q;
15            err = e0;
16          end
17        end
18      end
19    end
20    myPi = pbest/qbest;
21    return err
22  end
```

Πολυπλοκότητα: $\mathrm{O}(M^2)$

Η ακρίβεια του αλγορίθμου:

n	Τάξη σφάλματος
100	10^{-4}
1,000	10^{-8}
10,000	10^{-8}
100,000	10^{-9}
1,000,000	10^{-12}

Οι επιδόσεις των δύο αλγορίθμων:

n	*Matlab*	*Octave*	*Julia*
100	$0.002sec$	$0.24sec$	$0.001sec$
1,000	$0.02sec$	$24.1sec$	$0.01sec$
10,000	$1sec$	$47min$	$1.2sec$
100,000	$1min\ 32sec$	Est: $33h$	$1min\ 35sec$
1,000,000	$2h\ 34min$	N/A	$2h\ 46min$

3.2 Πιο σύντομος τρόπος

Επειδή ο παραπάνω τρόπος είναι πολύ αργός, χρειάζεται να περιορίσουμε τις τιμές των p και q που ελέγχουμε. Έτσι:

$$1 \leq q \leq ceil\left(\frac{M}{\pi}\right)$$

και $p = floor(q\pi)$ ή $p = ceil(q\pi)$

Ο κώδικας σε Matlab:

```
function err = pi_q_faster(M)
  pbest = 1;
  qbest = 1;
  err = abs((pbest/qbest-pi)/pi);
  for q = 1:ceil(M/pi)
    pMinus = floor(pi*q);
    errMinus = abs((pMinus/q-pi)/pi);
    pPlus = ceil(pi*q);
    errPlus = abs((pPlus/q-pi)/pi);
    if errMinus < errPlus
      p0 = pMinus;
      e0 = errMinus;
    else
      p0 = pPlus;
      e0 = errPlus;
    end
    if e0 < err
      err = e0;
      pbest = p0;
      qbest = q;
    end
  end
  myPi = pbest/qbest;
```

Και σε Julia:

```
function pi_q_faster(M)
  pbest = 1
  qbest = 1
  err = abs((pbest/qbest-pi)/pi)
  for q in 1:ceil(M/pi)
    pMinus = floor(pi*q)
    errMinus = abs((pMinus/q-pi)/pi)
    pPlus = ceil(pi*q)
    errPlus = abs((pPlus/q-pi)/pi)
    if errMinus < errPlus
```

```
11        p0 = pMinus
12        e0 = errMinus
13      else
14        p0 = pPlus
15        e0 = errPlus
16      end
17      if e0 < err
18        err = e0
19        pbest = p0
20        qbest = q
21      end
22    end
23    myPi = pbest/qbest
24    print("pbest = ", pbest, "\nqbest = ", qbest, "\n")
25    return err
26  end
```

Πολυπλοκότητα: $\mathrm{O}(M)$

Η ακρίβεια του αλγορίθμου:

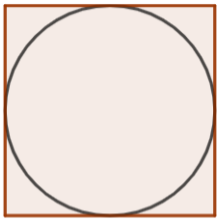
n	Τάξη σφάλματος
100,000	10^{-9}
1,000,000	10^{-12}
10,000,000	10^{-15}
100,000,000	10^{-16}

Οι επιδόσεις των δύο αλγορίθμων:

n	*Matlab*	*Octave*	*Julia*
100,000	$0.002sec$	$1.6sec$	$0.001sec$
1,000,000	$0.01sec$	$16sec$	$0.004sec$
10,000,000	$0.06sec$	$2min\ 40sec$	$0.03sec$
100,000,000	$0.6sec$	$28min$	$0.5sec$

4 Προσομοίωση Monte Carlo

Θεωρούμε κύκλο ακτίνας r και περιγεγραμμένο σε αυτόν τετράγωνο. Λαμβάνουμε n τυχαία σημεία εντός του τετραγώνου.



Έστω $hits$ τα σημεία που βρίσκονται μέσα στον κύκλο. Τότε:

$$\frac{hits}{n} \approx \frac{\pi r^2}{(2r)^2}$$

Άρα:

$$\pi \approx 4\frac{hits}{n}$$

Ένα σημείο θεωρείται ότι βρίσκεται εντός του κύκλου, όταν η απόστασή του από το κέντρο $(0, 0)$ είναι μικρότερη του r, δηλαδή

$$\sqrt{x^2 + y^2} \leq r$$

Ο κώδικας σε Matlab:

```
function err = pi_Monte_Carlo(n)
  hits = 0;
  for i = 1:n
    c = 2*rand(2,1)-1;
    if sqrt(c(1)*c(1) + c(2)*c(2)) <= 1
      hits = hits + 1;
    end
  end
  myPi = 4*hits/n;
  err = abs((pi-myPi)/pi);
```

Και σε Julia:

```
function pi_Monte_Carlo(n)
  hits = 0
  for i in 1:n
    x = 2*rand()-1
    y = 2*rand()-1
     if sqrt(x*x + y*y) <= 1
       hits = hits + 1
```

```
8       end
9     end
10    myPi = 4*hits/n
11    err = abs((pi-myPi)/pi)
12    return err
13  end
```

Πολυπλοκότητα: O(n)

Η ακρίβεια του αλγορίθμου:

n	Τάξη σφάλματος
100,000	10^{-4}
1,000,000	10^{-4}
10,000,000	10^{-4}
100,000,000	10^{-5}
1,000,000,000	10^{-6}

Οι επιδόσεις των δύο αλγορίθμων:

n	*Matlab*	*Octave*	*Julia*
100,000	$0.03sec$	$2.7sec$	$0.001sec$
1,000,000	$0.35sec$	$29sec$	$0.01sec$
10,000,000	$3.5sec$	$4min\ 53sec$	$0.1sec$
100,000,000	$35sec$	$47min$	$1.2sec$
1,000,000,000	$6min\ 15sec$	N/A	$11sec$

5 Συμπεράσματα

Όσον αφορά τους αλγορίθμους καθεαυτούς, παρατηρούμε ότι η μέθοδος της εύρεσης των εμβαδών των πολυγώνων είναι η πιο αποτελεσματική έχοντας και τους καλύτερους χρόνους. Αντίθετα, η εισαγωγή της τυχαιότητας (Monte Carlo) κάνει τα πράγματα πιο χρονοβόρα, ενώ παράγει το πιο ανακριβές αποτέλεσμα. Επίσης, στην ρητή προσέγγιση η ακρίβεια είναι πολύ καλή, αλλά απ΄ ό,τι φαίνεται απαιτείται η πιο γρήγορη μέθοδος, μιας και η brute force είναι πολύ αργή όσο αυξάνονται οι επαναλήψεις (έχει πολυπλοκότητα $O(M^2)$).

Συγκρίνοντας όμως τις δύο γλώσσες, Julia και Matlab, παρατηρούμε πολύ υψηλές ταχύτητες στις εκτελέσεις της πρώτης, γεγονός που μας επιτρέπει δοκιμές με μεγαλύτερες τιμές. Τέλος, το Matlab καταγράφει πολύ καλύτερους χρόνους από το Octave.