Oblig 3 i INF1000
Høsten 2005

Beregning av π med svært mange desimaler

Veiledning gitt 15. oktober 2005

Arnt Inge Vistnes
a.i.vistnes@fys.uio.no

Utgangspunkt:

1. John Machin's formel fra 1706:

$$\pi = 16 \arctan\frac{1}{5} - 4 \arctan\frac{1}{239}$$

2. Rekkeutvikling for arcustangens-funksjonen:

$$\arctan x = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \frac{1}{9}x^9 - \ldots$$

Arctan-funksjonen kan også skrives slik:

$$\arctan\frac{1}{x} = L_0 - L_1 + L_2 - \ldots + (-1)^{i-1}L_{i-1} + (-1)^i L_i + (-1)^{i+1}L_{i+1} + \ldots$$

hvor de ulike leddene kan beregnes rekursivt slik:

$$L_0 = \frac{1}{x} \quad \text{og} \quad L_i = L_{i-1}\,\frac{2i\text{-}1}{(2i+1)*x\text{^}2} \quad \text{for } i > 0$$

Når i øker, vil leddene L_i stadig bli mindre, såfremt at $x > 1$. Jo større x, jo raskere vil leddene avta.

En nyttig iaktagelse:

Siden

$$\arctan\frac{1}{x} = L_0 - L_1 + L_2 - \ldots$$

vil

$$Y * \arctan\frac{1}{x} = YL_0 - YL_1 + YL_2 - \ldots$$

Ved å la $L_0 = Y/x$ i stedet for $1/x$, vil automatisk alle videre ledd i rekursjonsformelen bli av typen YL_i. Det er altså unødvendig å mulitiplisere rekken *etter* at den er dannet, man kan gjøre det helt fra starten av!

Ved beregning av arctan vil det være bruk for ett tall hvor summen av alle L_i-ene lagres etter hvert som stadig nye ledd legges til. I min veiledning kaller jeg dette tallet for "*rekke*".

Det vil også være bruk for et tall som lagrer akkurat den L_i-verdien vi har i øyeblikket. Den verdien skal vi siden multiplisere med (2i-1) og dividere med (2i+1) (*etter* at *i* er oppjustert med én siden L_i-verdien forrige gang ble beregnet) for å finne den *nye* L_i-verdien. I min veiledning kaller jeg dette tallet for "*ledd*".

Problem:

Vi kan ikke bruke Javas innebygde *Math.atan*( )-funksjon siden den har en svært begrenset nøyaktighet, sett i vår sammenheng.

Vi kan heller ikke bruke *double* variable, igjen fordi disse har en svært begrenset nøyaktighet.

Vi må i stedet lage oss egne metoder for å håndtere tall med svært mange sifre, og vi må selv initiere disse tallene på en velvalgt måte.

Et godt valg i vår sammenheng er å basere seg på heltallsarimtetikk, siden den er eksakt! Selv divisjon er eksakt dersom vi tar vare på restleddet i divisjonen (modulus) på en eksakt måte.

Vi vil derfor lage et langt tall ved å bruke en integer array hvor *vi selv* velger at det bare skal finnes tall med fire sifre (tall i intervallet fra og med 0 til og med 9999). Vi lager da på sett og vis en tallrepresentasjon der grunntallet er 10000 i stedet for vanligvis 10. Her er et eksempel:

Et tall basert på syv integers, hver kan ha fire sifre:

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]
0003	2359	4286	6356	1886	0032	9989

Merk at i denne representasjonen finnes ikke noe desimalkomma. Det må vi holde rede på manuelt!

I det hele tatt, vi må selv initiere slike tall korrekt, for det er bare vi (ikke Java) som forstår hvordan vi vil håndtere tallene. Det er også vi som må lage metodene der vi kan summere, subtrahere, multiplisere og dividere med den nøyaktigheten vi selv velger.

Eksempel på initiering av et tall, dersom tallet skal være 1 og tallet skal ha 20 sifre *etter* desimalkommaet:

```
int [] tall;
tall = new int[6];
tall[0] = 1;
```

Først sier vi at `tall` skal være en array av heltall. Deretter sier vi at vi skal ha seks element i arrayen, og i `new`-kommandoen vet vi da at alle heltallene settes lik 0. Vi setter inn tallet 1 i det første. Re-

sultatet blir da:

tall[0]	tall[1]	tall[2]	tall[3]	tall[4]	tall[5]
0001	0000	0000	0000	0000	0000

Desimalkommaet finnes ikke i maskinens representasjon av tallet, men *vi selv* har valgt at det effektivt skal finnes mellom første og andre heltall i arrayen. Vi måtte derfor ha fem heltall *etter* kommaeet for å få 20 sifre etter komma.

Dersom vi opererer med flere lange tall, er det en stor fordel å la plassen for kommaet ligge fast og nøyaktig på samme måte for alle de lange tallene.

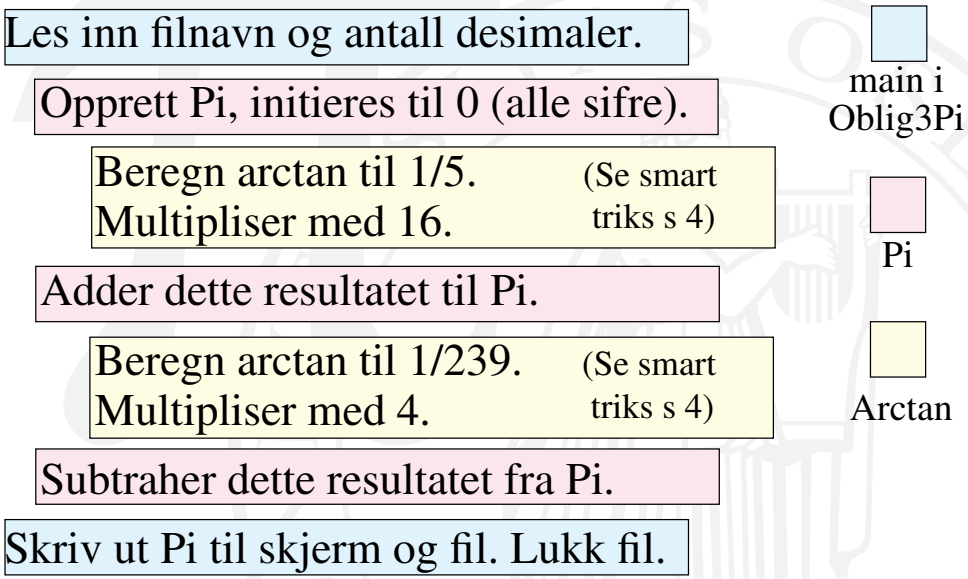
Hvordan skal vi ellers bygge opp programmet?

Machin's formel ser slik ut:

$$\pi = 16 \arctan\frac{1}{5} - 4 \arctan\frac{1}{239}$$

Oppgaveteksten krever at vi lager et objekt Pi som etter tur oppretter to objekter der arctan beregnes. Pi må ha ett langt tall der resultatet lagres. Arctan-objektet må også ha et langt tall som lagrer arctan. I tillegg må arctan ha enda et langt tall for å holde rede på leddene i rekkeutviklingen. Vi trenger derfor alt i alt tre lange tall!

Tankegangen blir da (omtrentlig) som følger:



Av dette ser vi at:

Klassen Oblig3Pi (med main):

Må ta seg av input av filnavn og antall desimaler.

Sjekke at input var korrekt (gjør det enkelt!).

Opprette et objekt av typen Pi, og samtidig overføre filnavn og antall desimaler til dette.

Skrive ut objektet pi på skjerm og fil ved å bruke en metode innenfor klassen Pi.

Klassen Pi:

Hente inn filnavn og antall desimaler (fra main).

Må initiere lang-tall-arrayen pi og opprette andre variable man har bruk for.

Må opprette arctan-objekt, med input "vinkel" og "faktor". To ganger.

Må addere eller subtrahere arctan-langt-tall til pi (og trenger da metoder for dette).

Må ha en metode for utskrift.

Klassen Arctan:

Hente inn "vinkel" og "faktor".

Må initiere lang-tall-arrayene *rekke* og *ledd*, og opprette andre variable man har bruk for.

Må ha metoder for å addere, subtrahere *ledd*-langt-tall til *rekke* og metoder for multiplikasjon og divisjon av *ledd*- og *rekke*-lange-tall.

Må ha en mekanisme som finner ut hvor lenge man må addere nye ledd i rekken (hvor lenge *ledd* er større enn 0 innenfor det antall sifre vi regner på).

Merk:

Hver gang man skal addere, subtrahere osv er det mulig å legge resultatet tilbake til ett av de eksisterende lang-tallene man jobber med. Man trenger derfor ikke å opprette enda flere lang-tall enn de tre man har.

Hovedlinjer i koden:

Det er svært viktig at man i oblig3 bruker objekter, og siden vi ikke har hatt så mange oppgaver tidligere der vi har brukt objekter aktivt, er nok noen usikre på hvordan filen da bygges opp. Her er en skisse av hvordan Oblig3Pi.java kan se ut:

```
class Oblig3Pi {
   public static void main(String[] args) {
       // Håndtering av input-parametre m.test
       // Koden må du selv lage selvfølgelig
        Pi p = new Pi (filnavn, antallSifre);
        p.skrivUt();
   }
}
```



```
class Pi {
   // Først deklareres objekt og
   // klassevariablene
   int [] pi; // peker til lang-tall-array
   int antallSifre;
   int antHeltall;
   ...  //andre deklarasjoner av obj.variable
   Out fil;
   static Out skjerm = new Out();

   //Så kommer konstruktøren i klassen Pi
   Pi (String filnvn, int nSifre) {
      fil = new Out(filnvn);
      antallSifre = nSifre;
      antHeltall = antallSifre/4 + 5;
      pi = new int[antHeltall]; // NB!
      ... // evt andre ting som må gjøres
```



```
      leggTilPi(new Arctan(antHeltall,16,5));
      trekkFraPi(new Arctan(antHeltall,4,239));
   }  // Slutt, konstruktør for Pi

   // Heretter følger metodene i Pi
   void leggTilPi(Arctan a) {
      // Her følger summasjons-metoden, og
      // leddene som summeres i løkka er
      // pi[i] og a.rekke[i]
   }

   void trekkFraPi(Arctan a) {
      // Tilsvarende her, kode for å trekke
      // a.rekke[i] fra pi[i]
   }
```



```
   void skrivUt( ) {
      // Utskrift-metode, bruker lagString(x)
      // Huske fil.close(); til slutt!
   }

   String lagString(int a) {
      // Se oblig-tekst s 5
   }
}  // Her er klassen Pi ferdig!
```

```
class Arctan {

   // Så kommer objektvariablene for arctan
   int antHeltall;
   int[] rekke;
   int[] ledd;
   ...  // andre variable man trenger
```

```
   // Så kommer konstruktøren for Arctan
   Arctan(int ntall, int faktor, int vinkel){
      antHeltall = ntall;
      rekke = new int [antHeltall];
      // osv
   } // Konstruktør for Arctan slutt

   // Heretter følger alle objektmetodene til
   // Arctan, i det minste metoder for hver
   // av de fire regneartene.

} // Slutt på klassen Arctan
```


Noen konkrete tips:

1. Skriving til fil:

Utskriftsobjekt må genereres:

```
Out fil;  // Før konstruktøren i Pi
fil = new Out(filnavn); // inni ds
....
fil.out(" " + pi[i]);  // i løkke
....
fil.close();              // til slutt
```

Her er `filnavn` en String med filnavnet. Det kan lønne seg å lage linjeskift etter f.eks. hvert 10. fire-sifrede tall, evt enda noe oppdeling for å finne fram.

2. Hente inn filnavn og antall sifre:

Parametrene gis allerede idet du starter program-met, f.eks. slik *java Oblig3Pi minfil.txt 10000*. I programmet hentes disse ut igjen ved:

```
public static void main(String[]
args) {
String filnavn = args[0];
int n = Integer.parseInt(args[1]);
```

Her er `n` antall sifre.

3. Én mulig addisjonsmetode:

Dersom vi skal addere *ledd* til *rekke*, gjør vi det ved å addere heltall på samme posisjon i arrayene med hverandre, resultatet legges inn i *rekke*:

```
rekke[i]=rekke[i]+ledd[i];
```

Problemet er at summen lett kan bli større enn 9999. På den annen side blir summen aldri større enn 19999. For å presse resultatet inn til fire siffer pr heltall i rekken, må vi derfor skrive:

```
if (rekke[i]>9999)rekke[i]-= 10000;
```

... men nå må vi legge menten (tallet 1) til det foran-gående tallet i arrayen. For at dette skal fungere tilfredsstillende, bør vi starte *bakfra* når vi legger sammen lang-tallene. Hele addisjonen kan derfor se slik ut (metode innenfor Arctan):

```
void addisjon( ) {
  int mente = 0;
  for (int i=nHeltall-1;i>=0;i--) {
  rekke[i] = rekke[i]+ledd[i]+mente;
  if (rekke[i]>9999) {
    rekke[i]-=10000;
    mente = 1;
  } else {mente = 0;}
}
```


Det bør bemerkes at *vi vet* at vi *i vår oblig-sammen-heng* ikke får mente ved addisjon av tallene foran desimalkommaet (i `rekke[0]`), siden dette tallet alltid vil holde seg godt under 10.

Av den grunn kan vi droppe å gjøre noe test på om noe slikt forekommer.

Det betyr at addisjonsmetoden vår ikke er 100 % generell for *alle mulige* tall bygd opp slik vi gjør. Men den holder for vårt formål!

4. Hva så med subtraksjon?

Man kan gå fram nesten som for addisjon, men man må teste hvorvidt tallet man får etter å ha subtra-hert to heltall er *mindre* enn null. I så fall må man *legge til* 10000 og la menten bli -1.

Så lenge man arbeider bakenfra, vil dette gå bra, fordi *vi vet* at vi alltid vil ende opp med et positivt tall til slutt og at vi derfor ikke får noe mente = -1 i siste subtraksjon.

Merk at så lenge testen om vi får et tall mindre enn null skjer på riktig sted i metoden, behøver vi ikke å bekymre oss om hvorvidt det er nødvendig å låne i flere ledd framover. Det tar metoden seg av helt av seg selv, uten ekstra tester, nettopp fordi vi ar-beider oss bakenfra og framover. Koden blir derfor behagelig kort.

5. Multiplikasjon

Når vi skal beregne neste ledd i arctan-rekken, vil vi ved 10000 desimaler kunne ha behov for å multiplisere et langt tall (ledd) med et tall som er bortimot (2i-1), dvs ca 2*10000 stort (i praksis ca 70 % mindre).

Hvert heltall innen lang-tallet er maksimalt 9999 stort.

Det betyr at produktet mellom disse maksimalt blir 200 000 000 stort.

Et heltall i Java kan maksimalt være 2 147 483 648.

Det betyr at vi har litt å gå på, men ikke mye.

Har vi f.eks. at *ledd[i]* er 9999, og vi skal multiplisere dette med (gjør det enkelt) 20 000, vil resultatet bli 199 980 000.

Bare siste fire sifrene skal lagres i ledd[i]. Disse sifrene kan vi plukke ut ved å bruke modulus-funksjonen:

```
ledd[i] = produkt % 10000;
```

Men menten blir nå svært stor:

```
mente = produkt / 10000;
```


I vårt tilfelle blir mente 19998

Denne skal i løkken legges til produktet vi får *etter* at tallet foran (*ledd[i-1]*) er multiplisert med faktoren (her 20 000).

Følger vi denne tankegangen litt videre, ser vi at vi nok kan gå opp til bortimot 40 000 sifre før vi får problemer med å foreta en enkel multiplikasjon mellom heltallene i ledd[ ] og de faktorene som inngår når man skal beregne stadig flere ledd i arctan-beregningen.

I vår obligsammenheng *holder det* med 10 000 sifre.

Et mulig multiplikasjonsprogram kunne da se slik ut:

```
void multiplikasjon(int faktor) {
      int mente = 0;
      for (int i=nHeltall-1;i>=0;i--) {
      produkt = (ledd[i]*faktor)+mente;
      ledd[i] = produkt % 10000;
      mente = produkt/10000;
}
```

Merk at dette er en nødløsning. Her har vi ikke brukt multiplikasjonsreglene fra barneskolen fullt ut, der "menten" brer seg over flere array-elementer. Arne Maus ønsker sterkt at dere forsøker å lage en mer generell metode, men det gjør dere om dere får tid.

En ny "vri-seg-litt-unna-løsning" er å bruke long variable (i stedet for integer) i multiplikasjon, modulus og divisjon internt i metoden ovenfor. Derved kan man bruke metoden opp til flere millioner sifre, selv om den da går litt tregt.

En mer generell metode er å implementere barneskole-algoritmen slik at dere kan håndtere multiplikasjon mellom to vilkårlig store tall. I så fall må det inn en ekstra løkke, men som sagt, dette er for finesmackere. Ta tak i dette om du har lyst og tid, og bruk den forenklede løsningen ellers.

6. Divisjon

Ved divisjon går vi fram omtrent som for multiplikasjon, men nå er det en klar fordel å arbeide *fra* de *mest* signifikante sifrene og *bakover*. Det blir altså en i++ i løkken i stedet for i--.

Nå blir prosedyren basert på:

```
forDeling=ledd[i]+tilOvers*10000;
ledd[i] = forDeling / dividend;
tilOvers = forDeling % dividend;
```


Denne kodesnutten kan kanskje virke forvirrende, men dersom du setter deg ned og lager et enkelt eksempel på divisjon, vil du forhåpentligvis skjønne hvordan det fungerer.

Dersom du sliter og ikke får det til, får du kontakte enten Tage (i gruppe 6 og 7) eller meg f.eks. via e-mail.

MERK:

Programsnuttene jeg har gitt for regneartene må tilpasses den sammenhengen de inngår i. F.eks. skal man innen Arctan-klassen bruke multiplisering både på *ledd*-tallet og på *rekke*-tallet.

Addisjon og subtraksjon skal også forekomme i to ulike sammenhenger, nemlig både i klassen Arctan (når *ledd* legges til eller trekkes fra *rekke*), og i klassen Pi (for å legge til eller trekke fra hele arctan-ledd til lang-tallet pi).

Et sidesprang:

Det kan være nyttig å lage noen lange tall kunstig som man kan teste aritmetikk-metodene på ved utvikling av programmet. Jeg selv syntes i alle fall det var svært nyttig. Jeg brukte en randomgenerator for å lage tall mellom 0 og 10000 (10000 ikke inkludert). Jeg kunne da lage f.eks. to lange tall og legge dem sammen, subtrahere, multiplisere et med et heltall, eller dividere og sjekke at alt gikk bra.

En sjekk på at divisjonen fungerer er forøvrig å sjekke at 1/239 kommer ut med sifre bakover så langt tallet vi har generert går.

Her er en kodesnutt som viser hvordan man kan generere et random langt tall og skrive det ut:

```
import java.util.*;
import easyIO.*;

class Oblig3testA {
  public static void main(String[] args) {
  In tast = new In();
  Out skjerm = new Out();
  int antallSifre = 100;
  int antallHeltall = antallSifre/4 + 4;
  int[] langtTall1 = new int[antallHeltall];
  int[] langtTall2 = new int[antallHeltall];
  java.util.Random randomGenerator = new java.util.
  Random();  //skulle vært én linje

  for (int i=0; i<antallHeltall; i++) {
     langtTall1[i] = randomGenerator.nextInt(10000);
```



```
     langtTall2[i] = randomGenerator.nextInt(10000);
  }
  // Tvinger første arrayelement til to nuller først:
  langtTall1[0] = langtTall1[0]/100;
  langtTall2[0] = langtTall2[0]/100;

  skjerm.out("Første lange tall: \n");
      for (int i=0; i<antallHeltall; i++) {
      skjerm.out(" " + utskrift(langtTall1[i]));
      // Ønsker linjeskift etter femten x 4 sifre
      if (((i+1) % 15) == 0) skjerm.outln();
      }

// Benytter meg av metoden gitt i obligteksten s 5:
public static String utskrift(int a) {
  String s = ""+a;
  while (s.length() < 4)
      s = "0" + s;
  return s;
}
}
```


Her er innrykkene ikke helt som de skal være, men hovedidéen med et slikt program går forhåpentligvis fram. Man kan starte med dette, og så lage etter tur de ulike metodene for aritmetikk man trenger, og teste at de fungerer. Så snart man er fornøyd med testingen, kan så metodene kopieres inn i Oblig3Pi-programmet (evt. etter at man har døpt om enkelte av variablene).

Det er ofte et godt tips i programmering å lage litt ekstra kode hvor man tester ut ulike deler av et program underveis, og ikke forsøke å lage det ønskede programmet fiks ferdig i en operasjon!

Forfininger:

Jeg har lagt min veiledning opp til en minimumsløsning. De som mestrer programmering bra, vil kunne forfine metodene og gjøre dem mer effektive. I den sammenheng vil jeg peke på to opplagte punkter der dette er aktuelt.

A: I beregningen av arctan skal annethvert ledd adderes og subtraheres. Man kan teste på `(-1)^i` eller `i % 2` for å se om man skal ha addisjon eller subtraksjon hver gang L_i beregnes. En bedre måte er å bestemme både L_i og L_{i+1} for *hver* runde i løkka, og oppdatere *i* med to (`i+=2`, ikke `i++`).

B: Ved beregningen av arctan blir L_i (dvs tallet *ledd*) mindre og mindre når *i* øker. Man **må** ha en mekanisme for å avslutte en løkke når dette tallet faktisk blir null innenfor vår nøyaktighet (antall sifre).

Underveis i beregningene er det sløsing med tid å f.eks. addere alle nullene foran i *ledd* til tallet som allerede eksisterer i *rekke*. Kan man holde rede på hvor mange nuller det er først i *ledd*, kan man avslutte adderingen (husk den skjer bakfra) når man kommer til nullene (forutsatt at menten da også er null), men...

Sløyf forfininger dersom du strever med å bli ferdig!

Husk forresten:

Du skal i obligen også vise hvordan man ved små modifikasjoner i programmet kan regne ut π med Carl Størmers metode. (S. levde 1874-1957, jobbet ved UiO)

Det var forresten en formel til Størmer som ble brukt da Shanks og Wrench i 1961 for første gang fikk bestemt π med over 100 000 sifre. De brukte 8 timer og 43 minutter på beregningen (på en IBM 7090 datamaskin i New York).

Forøvrig bruker selv en laptop i dag bare mellom 1 og 10 sek på å beregne π med 10 000 sifre i vår oblig, og koden kan typisk være ca 200 linjer langt.

Vel da står det bare igjen å si:

LYKKE TIL!

Håper du (i alle fall når du er ferdig) synes dette er en morsom oppgave!

Beregning av pi med 10008 desimaler

3. 1415 9265 3589 7932 3846 2643 3832 7950 2884 1971
6939 9375 1058 2097 4944 5923 0781 6406 2862 0899
8628 0348 2534 2117 0679 8214 8086 5132 8230 6647
0938 4460 9550 5822 3172 5359 4081 2848 1117 4502
8410 2701 9385 2110 5559 6446 2294 8954 9303 8196
4428 8109 7566 5933 4461 2847 5648 2337 8678 3165
2712 0190 9145 6485 6692 3460 3486 1045 4326 6482
1339 3607 2602 4914 1273 7245 8700 6606 3155 8817
4881 5209 2096 2829 2540 9171 5364 3678 9259 0360
0113 3053 0548 8204 6652 1384 1469 5194 1511 6094

3305 7270 3657 5959 1953 0921 8611 7381 9326 1179
3105 1185 4807 4462 3799 6274 9567 3518 8575 2724
8912 2793 8183 0119 4912 9833 6733 6244 0656 6430
8602 1394 9463 9522 4737 1907 0217 9860 9437 0277
0539 2171 7629 3176 7523 8467 4818 4676 6940 5132
0005 6812 7145 2635 6082 7785 7713 4275 7789 6091
7363 7178 7214 6844 0901 2249 5343 0146 5495 8537
1050 7922 7968 9258 9235 4201 9956 1121 2902 1960
8640 3441 8159 8136 2977 4771 3099 6051 8707 2113
4999 9998 3729 7804 9951 0597 3173 2816 0963 1859

5024 4594 5534 6908 3026 4252 2308 2533 4468 5035
2619 3118 8171 0100 0313 7838 7528 8658 7533 2083
8142 0617 1776 6914 7303 5982 5349 0428 7554 6873
1159 5628 6388 2353 7875 9375 1957 7818 5778 0532
1712 2680 6613 0019 2787 6611 1959 0921 6420 1989
3809 5257 2010 6548 5863 2788 6593 6153 3818 2796
8230 3019 5203 5301 8529 6899 5773 6225 9941 3891
2497 2177 5283 4791 3151 5574 8572 4245 4150 6959
5082 9533 1168 6172 7855 8890 7509 8381 7546 3746
4939 3192 5506 0400 9277 0167 1139 0098 4882 4012

8583 6160 3563 7076 6010 4710 1819 4295 5596 1989
4676 7837 4494 4825 5379 7747 2684 7104 0475 3464
6208 0466 8425 9069 4912 9331 3677 0289 8915 2104
7521 6205 6966 0240 5803 8150 1935 1125 3382 4300
3558 7640 2474 9647 3263 9141 9927 2604 2699 2279
6782 3547 8163 6009 3417 2164 1219 9245 8631 5030
2861 8297 4555 7067 4983 8505 4945 8858 6926 9956
9092 7210 7975 0930 2955 3211 6534 4987 2027 5596
0236 4806 6549 9119 8818 3479 7753 5663 6980 7426

5425 2786 2551 8184 1757 4672 8909 7777 2793 8000

8164 7060 0161 4524 9192 1732 1721 4772 3501 4144
1973 5685 4816 1361 1573 5255 2133 4757 4184 9468
4385 2332 3907 3941 4333 4547 7624 1686 2518 9835
6948 5562 0992 1922 2184 2725 5025 4256 8876 7179
0494 6016 5346 6804 9886 2723 2791 7860 8578 4383
8279 6797 6681 4541 0095 3883 7863 6095 0680 0642
2512 5205 1173 9298 4896 0841 2848 8626 9456 0424
1965 2850 2221 0661 1863 0674 4278 6220 3919 4945
0471 2371 3786 9609 5636 4371 9172 8746 7764 6575
7396 2413 8908 6583 2645 9958 1339 0478 0275 9009

9465 7640 7895 1269 4683 9835 2595 7098 2582 2620
5224 8940 7726 7194 7826 8482 6014 7699 0902 6401
3639 4437 4553 0506 8203 4962 5245 1749 3996 5143
1429 8091 9065 9250 9372 2169 6461 5157 0985 8387
4105 9788 5959 7729 7549 8930 1617 5392 8468 1382
6868 3868 9427 7415 5991 8559 2524 5953 9594 3104
9972 5246 8084 5987 2736 4469 5848 6538 3673 6222
6260 9912 4608 0512 4388 4390 4512 4413 6549 7627
8079 7715 6914 3599 7700 1296 1608 9441 6948 6855
5848 4063 5342 2072 2258 2848 8648 1584 5602 8506

0168 4273 9452 2674 6767 8895 2521 3852 2549 9546
6672 7823 9864 5659 6116 3548 8623 0577 4564 9803
5593 6345 6817 4324 1125 1507 6069 4794 5109 6596
0940 2522 8879 7108 9314 5669 1368 6722 8748 9405
6010 1503 3086 1792 8680 9208 7476 0917 8249 3858
9009 7149 0967 5985 2613 6554 9781 8931 2978 4821
6829 9894 8722 6588 0485 7564 0142 7047 7555 1323
7964 1451 5237 4623 4364 5428 5844 4795 2658 6782
1051 1413 5473 5739 5231 1342 7166 1021 3596 9536
2314 4295 2484 9371 8711 0145 7654 0359 0279 9344

0374 2007 3105 7853 9062 1983 8744 7808 4784 8968
3321 4457 1386 8751 9435 0643 0218 4531 9104 8481
0053 7061 4680 6749 1927 8191 1979 3995 2061 4196
6342 8754 4406 4374 5123 7181 9217 9998 3910 1591
9561 8146 7514 2691 2397 4894 0907 1864 9423 1961
5679 4520 8095 1465 5022 5231 6038 8193 0142 0937
6213 7855 9566 3893 7787 0830 3906 9792 0773 4672
2182 5625 9966 1501 4215 0306 8038 4477 3454 9202

6054 1466 5925 2014 9744 2850 7325 1866 6002 1324
3408 8190 7104 8633 1734 6496 5145 3905 7962 6856

1005 5081 0665 8796 9981 6357 4736 3840 5257 1459
1028 9706 4140 1109 7120 6280 4390 3975 9515 6771
5770 0420 3378 6993 6007 2305 5876 3176 3594 2187
3125 1471 2053 2928 1918 2618 6125 8673 2157 9198
4148 4882 9164 4706 0957 5270 6957 2209 1756 7116
7229 1098 1690 9152 8017 3506 7127 4858 3222 8718
3520 9353 9657 2512 1083 5791 5136 9882 0914 4421
0067 5103 3467 1103 1412 6711 1369 9086 5851 6398
3150 1970 1651 5116 8517 1437 6576 1835 1556 5088
4909 9898 5998 2387 3455 2833 1635 5076 4791 8535

8932 2618 5489 6321 3293 3089 8570 6420 4675 2590
7091 5481 4165 4985 9461 6371 8027 0981 9943 0992
4488 9575 7128 2890 5923 2332 6097 2997 1208 4433
5732 6548 9382 3911 9325 9746 3667 3058 3604 1428
1388 3032 0382 4903 7589 8524 3744 1702 9132 7656
1809 3773 4440 3070 7469 2112 0191 3020 3303 8019
7621 1011 0044 9293 2151 6084 2444 8596 3766 9838
9522 8684 7831 2355 2658 2131 4495 7685 7262 4334
4189 3039 6864 2624 3410 7732 2697 8028 0731 8915
4411 0104 4682 3252 7162 0105 2652 2721 1166 0396

6655 7309 2547 1105 5785 3763 4668 2065 3109 8965
2691 8620 5647 6931 2570 5863 5662 0185 5810 0729
3606 5987 6486 1179 1045 3348 8503 4611 3657 6867
5324 9441 6680 3962 6579 7877 1855 6084 5529 6541
2665 4085 3061 4344 4318 5867 6975 1456 6140 6800
7002 3787 7659 1344 0171 2749 4704 2056 2230 5389
9456 1314 0711 2700 0407 8547 3326 9939 0814 5466
4645 8807 9727 0826 6830 6343 2858 7856 9830 5235
8089 3306 5757 4067 9545 7163 7752 5420 2114 9557
6158 1400 2501 2622 8594 1302 1647 1550 9792 5923

0990 7965 4737 6125 5176 5675 1357 5178 2966 6454
7791 7450 1129 9614 8903 0463 9947 1329 6210 7340
4375 1895 7359 6145 8901 9389 7131 1179 0429 7828
5647 5032 0319 8691 5140 2870 8085 9904 8010 9421
1472 2131 7947 6477 7262 2414 2548 5454 0332 1541

8530 6142 2881 3758 5043 0633 2175 1829 7986 6223
7172 1591 6077 1669 2547 4873 8986 6549 4945 0114
6540 6284 3366 3937 9003 9769 2656 7214 6385 3067
3609 6571 2091 8076 3832 7166 4162 7488 8800 7869
2560 2902 2847 2104 0317 2118 6082 0419 0004 2296

6171 1963 7792 1337 5751 1495 9501 5660 4963 1862
9472 6547 3642 5230 8177 0367 5159 0673 5023 5072
8354 0567 0403 8674 3513 6222 2477 1589 1504 9530
9844 4893 3309 6340 8780 7693 2599 3978 0541 9341
4473 7744 1842 6312 9860 8099 8886 8741 3260 4721
5695 1623 9658 6457 3021 6315 9819 3195 1673 5381
2974 1677 2947 8672 4229 2465 4366 8009 8067 6928
2382 8068 9964 0048 2435 4037 0141 6314 9658 9794
0924 3237 8969 0706 9779 4223 6250 8221 6889 5738
3798 6230 0159 3776 4716 5122 8935 7860 1588 1617

5578 2973 5233 4460 4281 5126 2720 3734 3146 5319
7777 4160 3199 0665 5418 7639 7929 3344 1952 1541
3418 9948 5444 7345 6738 3162 4993 4191 3181 4809
2777 7103 8638 7734 3177 2075 4565 4532 2077 7092
1201 9051 6609 6280 4909 2636 0197 5988 2816 1332
3166 6365 2861 9326 6863 3606 2735 6763 0354 4776
2803 5045 0777 2355 4710 5859 5487 0279 0814 3562
4014 5171 8062 4643 6267 9456 1275 3181 3407 8330
3362 5423 2783 9449 7538 2437 2058 3531 1477 1199
2606 3813 3467 7687 9695 9703 0983 3913 0771 0987

0408 5913 3746 4144 2822 7726 3465 9470 4745 8784
7787 2019 2771 5280 7317 6790 7707 1572 1344 4730
6057 0073 3492 4369 3113 8350 4931 6312 8404 2512
1925 6517 9806 9411 3528 0131 4701 3047 8164 3788
5185 2909 2854 5201 1658 3934 1965 6213 4914 3415
9562 5865 8655 7055 2690 4965 2098 5803 3850 7224
2648 2939 7285 8478 3163 0577 7756 0688 8764 4624
8246 8579 2603 9535 2773 4803 0480 2900 5876 0758
2510 4747 0916 4396 1362 6760 4492 5627 4204 2083
2085 6611 9062 5454 3372 1315 3595 8450 6877 2460

2901 6187 6679 5240 6163 4252 2577 1954 2916 2991
9306 4553 7799 1403 7340 4328 7526 2888 9639 9587
9475 7291 7464 2635 7455 2540 7909 1451 3571 1136
9410 9119 3932 5191 0760 2082 5202 6187 9853 1887

7058 4297 2591 6778 1314 9699 0090 1921 1697 1737
2784 7684 7268 6084 9003 3770 2424 2916 5130 0500
5168 3233 6435 0389 5170 2989 3922 3345 1722 0138
1280 6965 0117 8440 8745 1960 1212 2859 9371 6231
3017 1144 4846 4090 3890 6449 5444 0061 9869 0754
8516 0263 2750 5298 3491 8740 7866 8088 1833 8510

2283 3450 8504 8608 2503 9302 1332 1971 5518 4306
3545 5007 6682 8294 9304 1377 6552 7939 7517 5461
3953 9846 8339 3638 3047 4611 9966 5385 8153 8420
5685 3386 2186 7252 3340 2830 8711 2328 2789 2125
0771 2629 4632 2956 3989 8989 3582 1167 4562 7010
2183 5646 2201 3496 7151 8819 0973 0381 1980 0497
3407 2396 1036 8540 6643 1939 5097 9019 0699 6395
5245 3005 4505 8068 5501 9567 3022 9219 1393 3918
5680 3449 0398 2059 5510 0226 3535 3619 2041 9947
4553 8593 8102 3439 5544 9597 7837 7902 3742 1617

2711 1723 6434 3543 9478 2218 1852 8624 0851 4006
6604 4332 5888 5698 6705 4315 4706 9657 4745 8550
3323 2334 2107 3015 4594 0516 5537 9068 6627 3337
9958 5115 6257 8432 2988 2737 2319 8987 5714 1595
7811 1963 5833 0059 4087 3068 1216 0287 6496 2867
4460 4774 6491 5995 0549 7374 2562 6901 0490 3778
1986 8359 3814 6574 1268 0492 5648 7985 5614 5372
3478 6733 0390 4688 3834 3634 6553 7949 8641 9270
5638 7293 1748 7233 2083 7601 1230 2991 1367 9386
2708 9438 7993 6201 6295 1541 3371 4248 9283 0722

0126 9014 7546 6847 6535 7616 4773 7946 7520 0490
7571 5552 7819 6536 2132 3926 4061 6013 6358 1559
0742 2020 2031 8727 7605 2772 1900 5561 4842 5551
8792 5303 4351 3984 4253 2234 1576 2336 1064 2506
3904 9750 0865 6271 0953 5919 4658 9751 4131 0348
2276 9306 2474 3536 3256 9160 7815 4781 8115 2843
6679 5706 1108 6153 3150 4452 1274 7392 4544 9454
2368 2886 0613 4084 1486 3776 7009 6120 7151 2491
4043 0272 5386 0764 8236 3414 3346 2351 8975 7664
5216 4137 6796 9031 4950 1910 8575 9844 2391 9862

9164 2193 9949 0723 6234 6468 4411 7394 0326 5918
4044 3780 5133 3894 5257 4239 9508 2965 9122 8508
5558 2157 2503 1071 2570 1266 8302 4029 2952 5220

1187 2676 7562 2041 5420 5161 8416 3484 7565 1699
9811 6141 0100 2996 0783 8690 9291 6030 2884 0026
9104 1407 9288 6215 0784 2451 6709 0870 0069 9282
1206 6041 8371 8065 3556 7252 5325 6753 2861 2910
4248 7761 8258 2976 5157 9598 4703 5622 2629 3486
0034 1587 2298 0534 9896 5022 6291 7487 8820 2734
2092 2224 5339 8562 6476 6914 9055 6284 2503 9127

5771 0284 0279 9806 6365 8254 8892 6488 0254 5661
0172 9670 2664 0765 5904 2909 9456 8150 6526 5305
3718 2941 2703 3693 1378 5178 6090 4070 8667 1149
6558 3434 3476 9338 5781 7113 8645 5873 6781 2301
4587 6871 2660 3489 1390 9562 0099 3936 1031 0291
6161 5288 1384 3790 9904 2317 4733 6394 8045 7593
1493 1405 2976 3475 7481 1935 6709 1101 3775 1721
0080 3155 9024 8530 9066 9203 7671 9220 3322 9094
3346 7685 1422 1447 7379 3937 5170 3443 6619 9104
0337 5111 7354 7191 8550 4644 9026 3655 1281 6228

8244 6257 5916 3330 3910 7225 3837 4218 2140 8835
0865 7391 7715 0968 2887 4782 6569 9599 5744 9066
1758 3441 3752 2397 0968 3408 0053 5598 4917 5417
3818 8399 9446 9748 6762 6551 6582 7658 4835 8845
3142 7756 8790 0290 9517 0283 5297 1634 4562 1296
4043 5231 1760 0665 1012 4120 0659 7558 5127 6178
5838 2920 4197 4844 2360 8007 1930 4576 1893 2349
2292 7965 0198 7518 7212 7267 5079 8125 5470 9589
0455 6357 9212 2103 3346 6974 9923 5630 2549 4780
2490 1141 9521 2382 8153 0911 4079 0738 6025 1522

7429 9581 8072 4716 2591 6685 4513 3312 3948 0494
7079 1191 5326 7343 0282 4418 6041 4263 6395 4800
0448 0026 7049 6248 2017 9289 6476 6975 8318 3271
3142 5170 2969 2348 8962 7668 4403 2326 0927 5249
6035 7996 4692 5650 4936 8183 6090 0323 8092 9345
9588 9706 9536 5349 4060 3402 1665 4437 5589 0045
6328 8225 0545 2556 4056 4482 4651 5187 5471 1962
1844 3965 8253 3754 3885 6909 4113 0315 0952 6179
3780 0297 4120 7665 1479 3942 5902 9896 9594 6995
5657 6121 8656 1967 3378 6236 2561 2521 6320 8628

6922 2103 2748 8921 8654 3648 0229 6780 7057 6561
5144 6320 4692 7906 8212 0738 8377 8142 3356 2823
6089 6320 8068 2224 6801 2248 2611 7718 5896 3814
0918 3903 6736 7222 0888 3215 1375 5600 3727 9839
4004 1529 7002 8783 0766 7094 4474 5601 3455 6417
2543 7090 6979 3961 2257 1429 8946 7154 3578 4687
8861 4445 8123 1459 3571 9849 2252 8471 6050 4922
1242 4701 4121 4780 5734 5510 5008 0190 8699 6033
0276 3478 7081 0817 5450 1193 0714 1223 3908 6639
3833 9529 4257 8690 5076 4310 0638 3519 8343 8934

1596 1318 5434 7546 4955 6978 1038 2930 9716 4651
4384 0700 7073 6041 1237 3599 8434 5225 1610 5070
2705 6235 2660 1276 4848 3084 0761 1830 1305 2793
2054 2746 2865 4036 0367 4532 8651 0570 6587 4882
2569 8157 9367 8976 6974 2205 7505 9683 4408 6973
5020 1410 2067 2358 5020 0724 5225 6326 5134 1055
9240 1902 7421 6248 4391 4035 9989 5353 9459 0944
0704 6912 0914 0938 7001 2645 6001 6237 4288 0210
9276 4579 3106 5792 2955 2498 8727 5846 1012 6483
6999 8922 5695 9688 1592 0560 0101 6552 5637 5678

((5667 2279 6619 8857 8279 4672))

Siste sifrene bare med for å være sikker på at ikke avrundingsfeil skal innvirke på de 10000 første sifrene etter komma.