

lab2ans

Calculus A, Lab 2 Solutions

Tim McLarnan

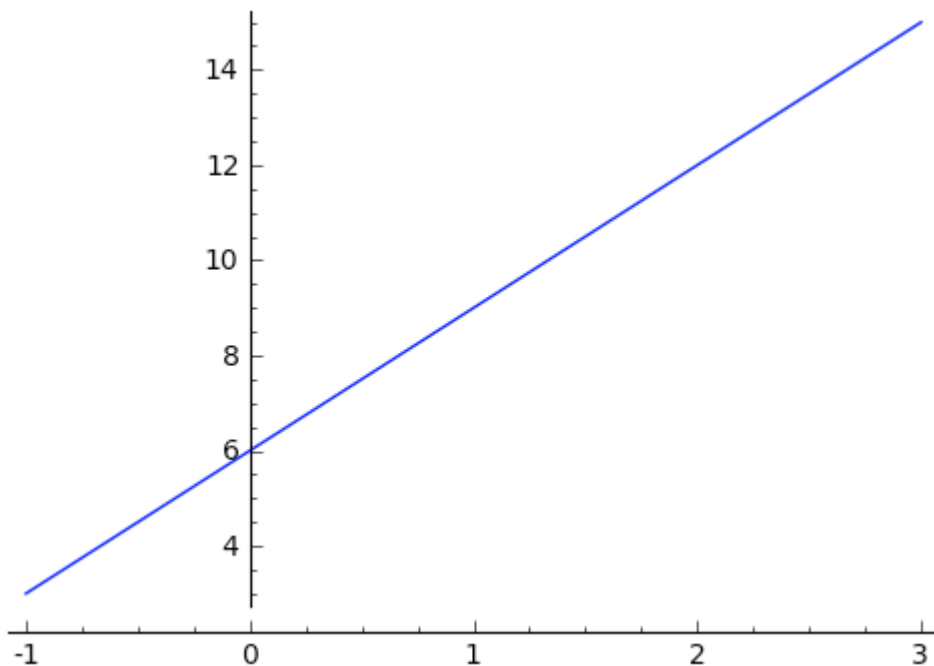
Problem 1

To investigate

$$\lim_{x \rightarrow 2} \frac{3x^2 - 12}{x - 2}$$

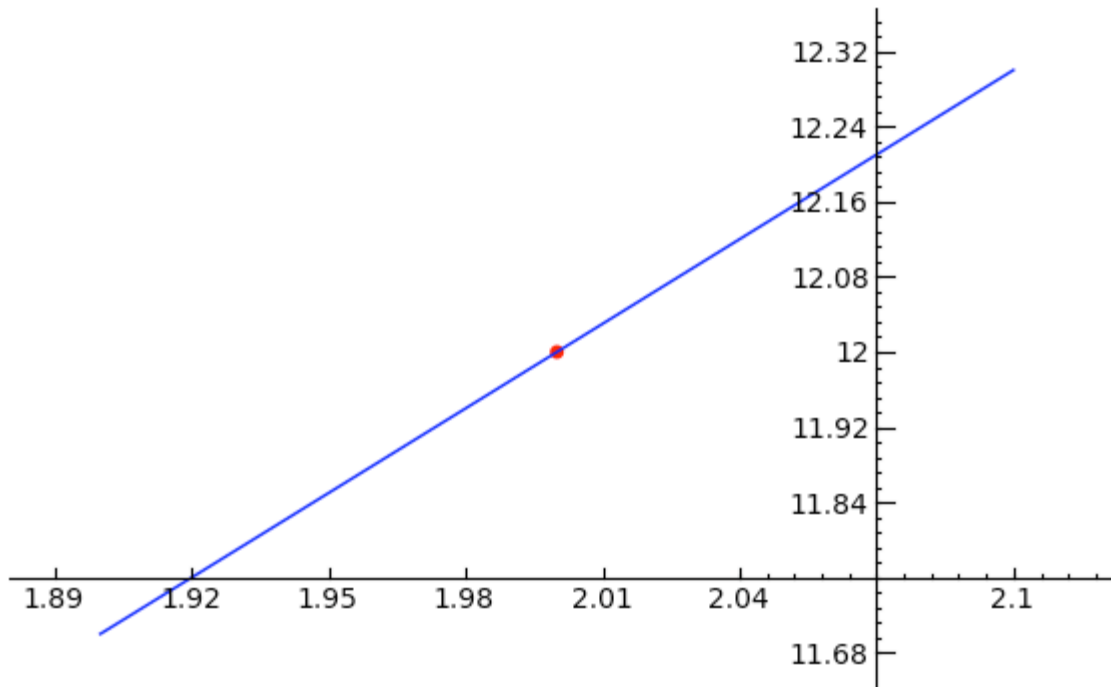
graphically, just plot the function near $x = 2$.

```
plot((3*x^2-12)/(x-2), (x, -1, 3))
```



It looks as though the limit is about 12. Here's a bit of confirmation: the same plot zoomed in and with a point at $(2, 12)$ added:

```
p1 = plot((3*x^2-12)/(x-2), (x, 1.9, 2.1))
dot = point((2,12),pointsize=20,rgbcolor='red')
show(p1+dot)
```



To get numerical evidence for the belief that the limit is 12, let's compute the value of the function

$$f(x) = \frac{3x^2 - 12}{x - 2}$$

at some points just above and just below $x = 2$.

$f(x) = (3x^2 - 12) / (x - 2)$
$f(1.9)$
11.7000000000000
$f(1.99)$
11.9699999999999
$f(1.999)$
11.9969999999996
$f(1.9999)$
11.9997000000018

So coming from below, each additional 9 in the value of x gives us one more 9 before the 7 (or the

$6\bar{9}$) in $f(x)$.

Similarly, as we come down from above, each additional 0 in the value of x gives us one more 0 before the 3 in $f(x)$.

$f(2.1)$

12.3000000000000

$f(2.01)$

12.0299999999999

$f(2.001)$

12.0030000000004

$f(2.0001)$

12.0002999999982

To me, this numerical evidence is much more persuasive than the graph. The graph only shows the function at one scale, but the simple pattern in the values of f at points closer and closer to 2 seems like something which is almost certain to persist at every scale. This evidence feels even more persuasive to me if instead of using approximate floating-point values for x we use exact fractions.

$f(21/10)$

$\frac{123}{10}$

$f(201/100)$

$\frac{1203}{100}$

$f(2001/1000)$

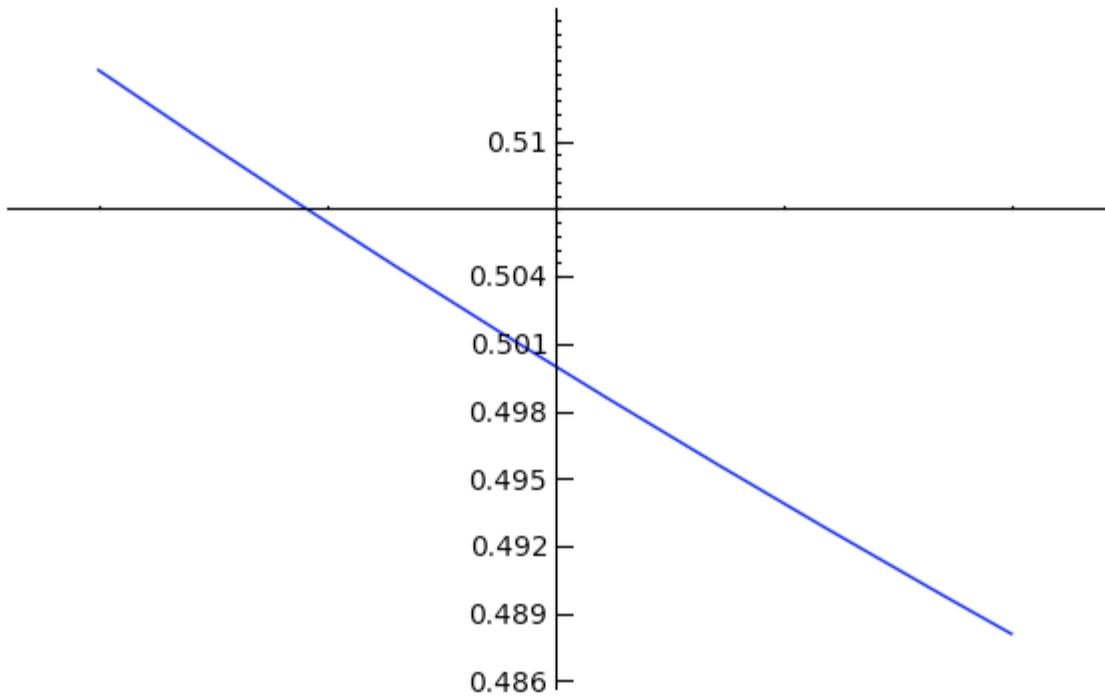
$\frac{12003}{1000}$

Problem 2

Graphical evidence makes it appear quite strongly that

$$\lim_{x \rightarrow 1} \frac{\sqrt{x} - 1}{x - 1} = \frac{1}{2}.$$

`plot((sqrt(x)-1)/(x-1), (x, 0.9, 1.1))`



Numerical evidence backs this up. Again, there are very visible patterns at least for the first few digits of the value of the function.

$$g(x) = (\sqrt{x} - 1) / (x - 1)$$

$$g(1.1)$$

0.488088481701516

$$g(1.01)$$

0.498756211208895

$$g(1.001)$$

0.499875062461019

$$g(1.0001)$$

0.499987500624021

$$g(0.9)$$

0.513167019494862

$$g(0.99)$$

0.501256289338003

```
g(0.999)
```

```
0.500125062539047
```

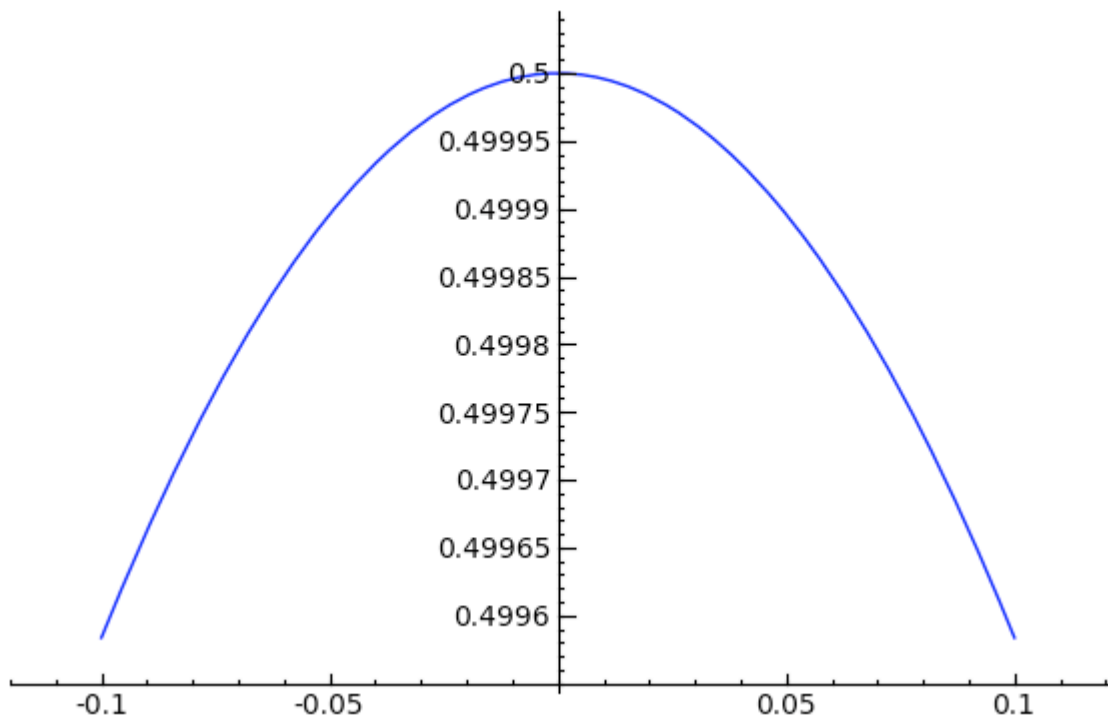
```
g(0.9999)
```

```
0.500012500624980
```

Finally, here's the evidence that

$$\lim_{x \rightarrow 0} \frac{1 - \cos(x)}{x^2} = \frac{1}{2}.$$

```
plot((1-cos(x))/x^2, (x, -0.1, 0.1))
```



```
k(x) = (1-cos(x))/x^2
```

```
k(0.1)
```

```
-100.000000000000 cos(0.100000000000000) + 100.000000000000
```

I'm not sure why *Sage* doesn't want to work this out all the way, but we can force it to do so by wrapping the result in an `n()`.

```
n(k(0.1))
```

```
0.499583472197415
```

```
n(k(0.01))
```

```
0.499995833346475
```

```
n(k(0.001))
```

```
0.499999958323315
```

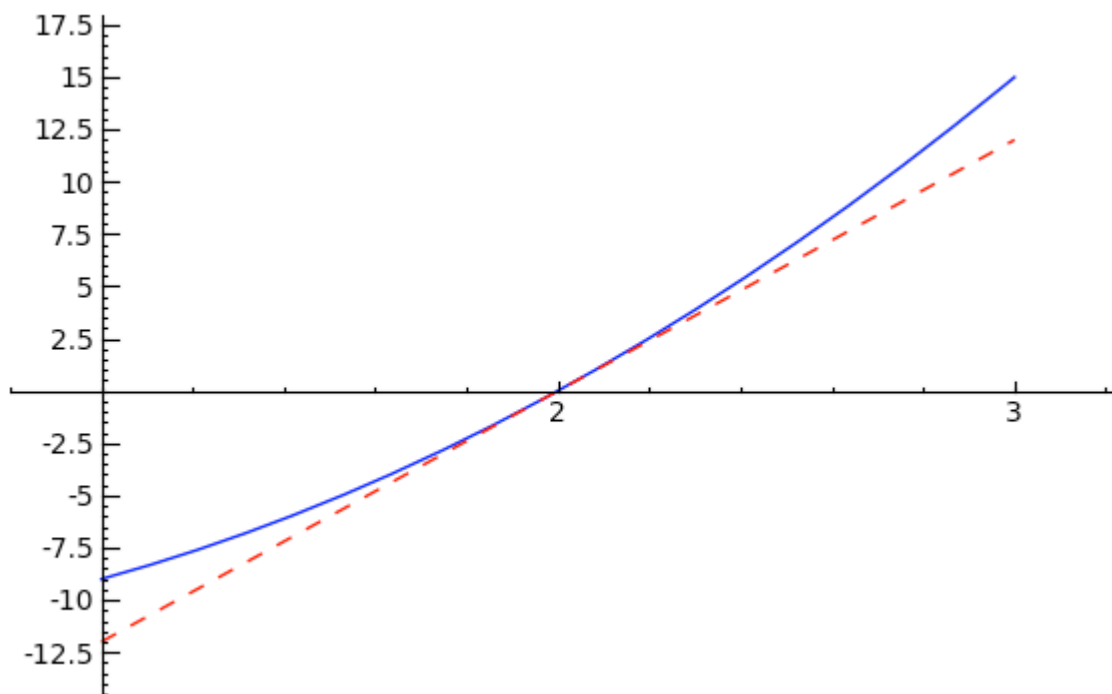
```
n(k(-0.001))
```

```
0.499999958323315
```

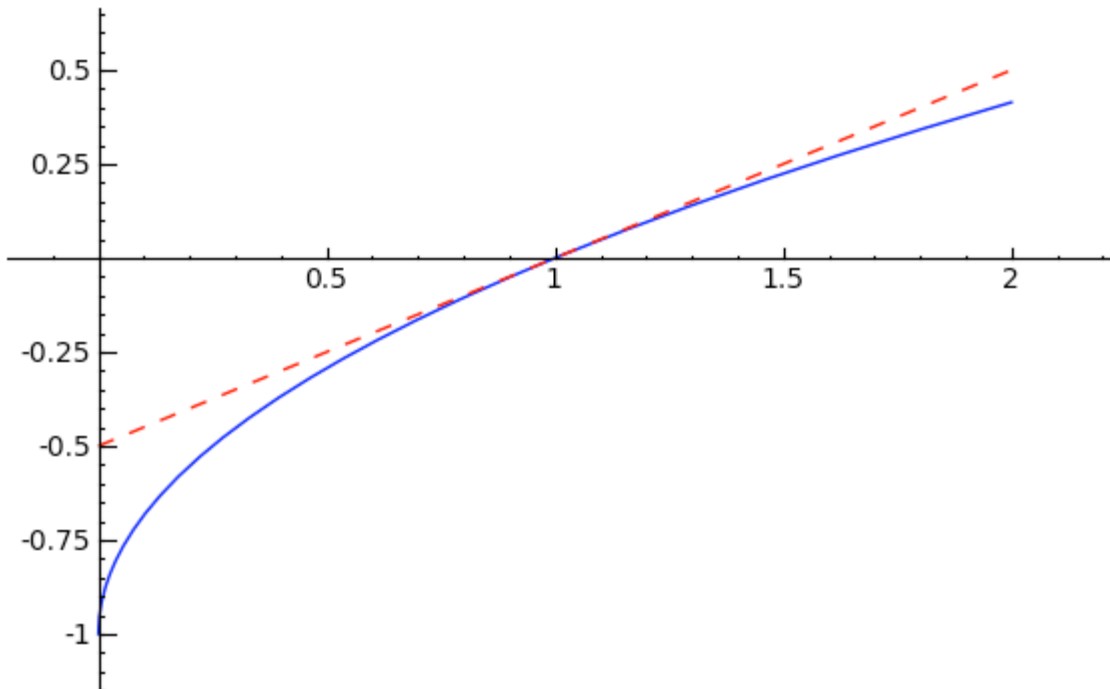
Problem 3

Here are the three plots:

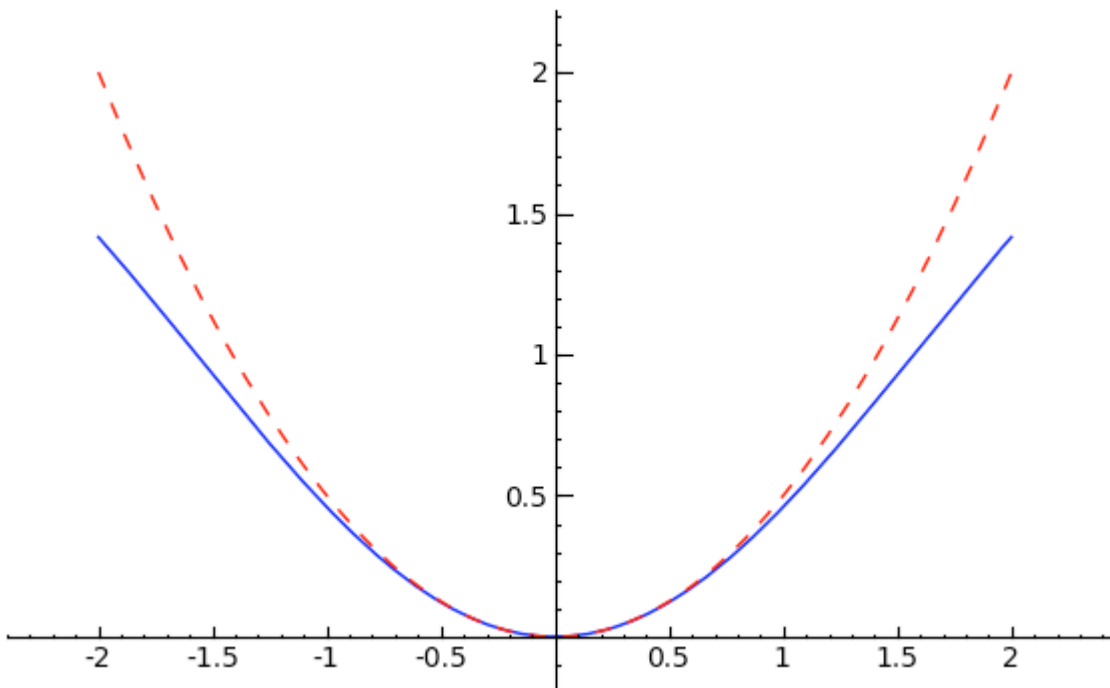
```
p1 = plot(3*x^2-12, (x, 1, 3), color='blue')
p2 = plot(12*(x-2), (x, 1, 3), color='red', linestyle='--')
show(p1+p2)
```



```
p1 = plot(sqrt(x)-1, (x, 0, 2), color='blue')
p2 = plot((1/2)*(x-1), (x, 0, 2), color='red', linestyle='--')
show(p1+p2)
```



```
p1 = plot(1-cos(x), (x, -2, 2), color='blue')
p2 = plot((1/2)*x^2, (x, -2, 2), color='red', linestyle='--')
show(p1+p2)
```



In all three plots, the pairs of curves go through the same point and are tangent to one another. Near the point in question, the curves are essentially equal.

Why should this be, and what does it have to do with the limits? Well, let's just look at the first case. We saw strong evidence earlier that

$$\lim_{x \rightarrow 2} \frac{3x^2 - 12}{x - 2} = 12.$$

This means that if x is close to 2, then

$$\frac{3x^2 - 12}{x - 2} \approx 12.$$

Another way to say this would be that if x is close to 2, then $3x^2 - 12 \approx 12(x - 2)$. This approximate equality is shown by the fact that the two curves lie atop one another when x is close to 2.

The other two examples work just the same way, interpreting the other limits in Problems 2 and 3.

Problem 4

As long as I believe that the shortest distance between 2 points is a straight line, of course the circumference of the circle is larger than the perimeter of the square.

The triangle in Figure 1 is isosceles; so by Pythagoras, $a_4 = b_4 = \sqrt{2}/2$. The perimeter of the square is therefore $8(\sqrt{2}/2) = 4\sqrt{2}$, which means that if we use half the perimeter of the square as an approximation for π we get $\pi \approx 2\sqrt{2} \approx 2.82842712474619$. The actual value of π is larger than this estimate.

```
(2*sqrt(2)).n()
```

2.82842712474619

Problem 5

Now comes the fun. The triangle in Figure 2 containing the label b_4 has horizontal side b_4 , vertical side $1 - a_4$, and therefore hypotenuse

$$2b_8 = \sqrt{b_4^2 + (1 - a_4)^2},$$

which means that

$$b_8 = \frac{1}{2} \sqrt{b_4^2 + (1 - a_4)^2}.$$

Given this, we can apply Pythagoras to the triangle having a_8 as one of its sides and get

$$a_8 = \sqrt{1 - b_8^2}.$$

Here's how *Sage* simplifies and evaluates these quantities:

```
a4 = sqrt(2)/2
b4 = a4
b8 = (1/2)*sqrt(b4^2+(1-a4)^2)
b8
```

$$\frac{1}{2} \sqrt{\frac{1}{4} (\sqrt{2} - 2)^2 + \frac{1}{2}}$$

```
b8 = b8.expand(); b8
```

$$\frac{1}{2} \sqrt{-\sqrt{2} + 2}$$

```
a8 = sqrt(1-b8^2); a8
```

$$\sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2}}$$

```
pi_approx_8 = 8*b8
pi_approx_8
```

$$4 \sqrt{-\sqrt{2} + 2}$$

```
pi_approx_8.n()
```

3.06146745892072

What happens now if we try to move from an octagon to a 16-gon (a hexkaidecagon, if you want to do it in Greek)? If you draw out one side of the octagon and the 2 sides of the 16-gon to which it gives rise, you get a figure very much like Figure 2. From this figure, it's easy to see that the equations connecting a_8 and b_8 to a_{16} and b_{16} are the same as the equations connecting a_4 and b_4 to a_8 and b_8 :

$$b_{16} = \frac{1}{2} \sqrt{b_8^2 + (1 - a_8)^2}, \quad a_{16} = \sqrt{1 - b_{16}^2}.$$

```
b16 = (1/2)*sqrt(b8^2+(1-a8)^2).expand()
b16
```

$$\frac{1}{2} \sqrt{-2 \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2}} + 2}$$

```
a16 = sqrt(1-b16^2); a16
```

$$\sqrt{\frac{1}{2} \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2}} + \frac{1}{2}}$$

```
pi_approx_16 = 16*b16
pi_approx_16
```

$$8 \sqrt{-2 \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2}} + 2}$$

```
pi_approx_16.n()
```

3.12144515225805

A couple of things are happening here. First, of course, we're getting closer and closer approximations to π . More than that, though, we're getting interesting approximations to π that seem to follow a pattern. A bit of algebra cleans up our approximations for the square, the octagon, and the 16-gon as

$$2\sqrt{2}$$

$$2^2 \sqrt{2 - \sqrt{2}}$$

$$2^3 \sqrt{2 - \sqrt{2 + \sqrt{2}}}.$$

Let's see if this pattern continues when we double the number of sides again to get a 32-gon. The equations we need to use are the same.

```
b32 = (1/2)*sqrt(b16^2+(1-a16)^2).expand()
b32
```

$$\frac{1}{2} \sqrt{-2 \sqrt{\frac{1}{2} \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2}} + \frac{1}{2}} + 2}$$

```
a32 = sqrt(1-b32^2); a32
```

$$\sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2}} + \frac{1}{2}} + \frac{1}{2}}$$

```
pi_approx_32 = 32*b32
pi_approx_32
```

$$16 \sqrt{-2 \sqrt{\frac{1}{2} \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2} + \frac{1}{2} + 2}}}$$

```
pi_approx_32.n()
```

3.13654849054594

Further, staring at our new approximation for π makes clear that it could be rewritten as

$$2^4 \sqrt{2 - \sqrt{2 + \sqrt{2 + \sqrt{2}}}}$$

continuing our pattern of approximations.

OK, now let's automate what we are doing and produce the approximations for π that we would get using a 64-gon, a 128-gon, a 256-gon, a 512-gon, and a 1024-gon.

```
a = a32
b = b32
for n in [6..10]:
    b = (1/2)*sqrt(b^2+(1-a)^2).expand()
    a = sqrt(1-b^2)
    print(2^n, (2^n*b).n())
```

```
(64, 3.14033115695474)
(128, 3.14127725093276)
(256, 3.14151380114415)
(512, 3.14157294036788)
(1024, 3.14158772527996)
```

```
2^10*b
```

$$512 \sqrt{-2 \sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{2} \sqrt{\frac{1}{4} \sqrt{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + 2}}}}}}}}}}$$

Good. So we're now getting approxiamtions to π that are accurate to 5 decimal places, and our pattern for the form of the approximants still holds.

Can we get more precision still? And how rapidly does this process converge? Well, let's try doubling the number of sides 10 more times until we're working with a ploygon with $2^{20} = 1,048,576$ sides. This time for each polygon I'll print the number of sides, the current

approximation for π , and the error of this approximation. (Of course, we only know this error exactly because we know the value of π . If this were the first accurate estimate of π , we wouldn't have these exact errors available.)

```

a = sqrt(2)/2
b = sqrt(2)/2
error = RealField(100)(4*b-pi)
for n in [3..20]:
    b = (1/2)*sqrt(b^2+(1-a)^2).expand()
    a = sqrt(1-b^2)
    approx = RealField(100)(2^n*b)
    olderror = error
    error = RealField(100)(approx-pi)
    print 2^n, approx, error, olderror/error
    print

```

```

8 3.0614674589207181738276798722 -0.080125194669075064634963511
3.9084526426052077500803632385

16 3.1214451522580522855725578956 -0.02014750133174095289008548
3.9769296127477370679981872958

32 3.1365484905459392638142580444 -0.00504416304385397464838533
3.9942208760062852555824175519

64 3.1403311569547529123171185244 -0.00126149663504032614552485
3.9985544976841958502906725237

128 3.1412772509327728680620197707
-0.00031540265702037040062361253167 3.9996385793251320178775457

256 3.1415138011443010763285150570
-0.000078852445492162134128326243450 3.999909642012575956144843

512 3.1415729403670913841358001119
-0.000019713222701854326843271340298 3.999977410326971482135082

1024 3.1415877252771597006288542581
-4.9283126335378337891252076884e-6 3.99999435257073203702919216

2048 3.1415914215111999739979717440
-1.2320785932644646716392917787e-6 3.99999858814199483129450355

4096 3.1415923455701177423403756862
-3.0801967549612226769706674448e-7 3.99999964703545569277220664

8192 3.1415925765848726656816065866
-7.7004920572781036796708033104e-8 3.99999991175886126565790269

```

```

16384 3.1415926343385629890954726054
-1.9251230249367170777890770700e-8 3.99999997793971393964209412

32768 3.1415926487769856694850541311
-4.8128075689775892521499549451e-9 3.99999999448488520467736844

65536 3.1415926523865913458034212033
-1.2032018926592221799939866616e-9 3.99999999862093013820088120

131072 3.1415926532889927652715243278
-3.0080047319111905546659764153e-10 3.9999999996500868474181926

262144 3.1415926535145931201661275286
-7.5200118296515854713203233043e-11 4.000000000672291045321260

524288 3.1415926535709932088826634709
-1.8800029579979912364527629345e-11 3.9999999987551192595021283

1048576 3.1415926535850932310929150475
-4.7000073697283357661620699501e-12 4.0000000215034916648909593

```

What these calculations show is that by using this method with a 10^{20} -gon, we can approximate π with an error of 4.7×10^{-12} , and that every time we double the number of sides of the polygon, we reduce the error by a factor of roughly 4.

Are there ways to use this idea to squeeze even more accuracy out of our calculations? Well, one thing we might try doing is using circumscribed polygons instead of inscribed ones. If we let A_n be the distance from the center of a unit circle to a vertex of a circumscribed regular n -gon, and let B_n be half the length of the side of this n -gon, then geometry rather like what we did with inscribed polygons shows that $A_4 = \sqrt{2}$, that $B_4 = 1$ and that when we double the number of sides,

$$B_{2n} = \frac{1}{2} \left(B_n - \frac{(A_n - 1)^2}{B_n} \right)$$

$$A_{2n} = \sqrt{1 + B_{2n}^2}.$$

Here are the approximations to π obtained from perimeters of circumscribed polygons with 2^n sides, $3 \leq n \leq 20$, together with the associated errors and the ratios by which the errors decrease when n replaces $n - 1$.

```

A = RealField(100)(sqrt(2))
B = RealField(100)(1)
error = RealField(100)(4*B-pi)

```

```

for n in [3..20]:
  B = (1/2)*(B-(A-1)^2/B)
  A = sqrt(1+B^2)
  approx = RealField(100)(2^n*B)
  olderror = error
  error = RealField(100)(approx-pi)
  print 2^n, approx, error, olderror/error
  print

```

```

8 3.3137084989847603904135097937 0.1721158453949671519508664104
4.9873812863676495279541381157

```

```

16 3.1825978780745281105855619623 0.041005224484734872122918579
4.1974125872433936277560921639

```

```

32 3.1517249074292560984703206813 0.010132253839462860007677298
4.0469993285234035659702362915

```

```

64 3.1441183852459042627419725614 0.002525731656111024279329178
4.0116113740538530693081019649

```

```

128 3.1422236299424568453862085070
0.00063097635266360692356512372136 4.00289431679790734281369874

```

```

256 3.1417503691689664591072136280
0.00015771557917322064457024469940 4.00072304823228083513102951

```

```

512 3.1416320807031818057187151879
0.000039427113388567256071804597655 4.0001807289030113604587360

```

```

1024 3.1416025102568089467636896585
9.8566670157083010462752202858e-6 4.000045180154036063432273905

```

```

2048 3.1415951177495890503530922360
2.4641597958118904488527089918e-6 4.000011294909034128054840230

```

```

4096 3.1415932696293073107894587868
6.1603951407232681540355440132e-7 4.000002823719166467311829744

```

```

8192 3.1415928075996445765282544360
1.5400985133806561105268364191e-7 4.000000705929285864593197015

```

```

16384 3.1415926920922543742284195572
3.8502461135765776173903204374e-8 4.000000176482289856661779647

```

```

32768 3.1415926632154084162321791901
9.6256151777695358068099178230e-9 4.000000044120570488573477933

```

```

65536 3.1415926559961970262692831628
2.4064037878066397794924650566e-9 4.000000011030142498667330215

```

```

131072 3.1415926541913941849995693188
6.0160094653692593555671196057e-10 4.00000000275753561694868881

262144 3.1415926537401934750709539916
1.5040023660831060832844848464e-10 4.00000000068938390378296002

524288 3.1415926536273932976131009806
3.7600059150457597356756210092e-11 4.00000000017234597625213521

1048576 3.1415926535991932532501565292
9.4000147875131459181147509826e-12 4.00000000004308649437820256

```

Again, the errors shrink by roughly a factor of 4 every time we double the number of sides.

It's also nice that the correct value of π lies in between the estimates we're getting with inscribed and circumscribed polygons. Thus, even if we have no other method for computing π , we now know that

$$3.141592653585 < \pi < 3.141592653600.$$

Pretty good, eh?

There's another thing to be noticed if we stare at our two tables of numbers, though: the estimate I_n obtained from inscribed n -gons is always too small, the estimate C_n obtained using circumscribed n -gons is always too large, and the error with circumscribed polygons is very nearly twice as large numerically as the error with inscribed polygons. This suggests that we'd get better estimates still if we used the point

$$\frac{2I_n + C_n}{3}$$

that lies a third of the way from the inscribed estimate to the circumscribed estimate. (In Calc B, we'll do a very similar thing to approximate areas using something called Simpson's Rule.)

Here are the approximations, errors, and factors by which the error decreases using this new weighted average.

```

a = RealField(100)(sqrt(2)/2)
b = RealField(100)(sqrt(2)/2)
A = RealField(100)(sqrt(2))
B = RealField(100)(1)
approx = 4*(2*b+B)/3
error = RealField(100)(approx-pi)
for n in [3..20]:
    b = (1/2)*sqrt(b^2+(1-a)^2)

```

```

a = sqrt(1-b^2)
B = (1/2)*(B-(A-1)^2/B)
A = sqrt(1+B^2)
approx = RealField(100)(2^n*(2*b+B)/3)
olderror = error
error = RealField(100)(approx-pi)
print 2^n, approx, error, olderror/error
print

```

```

8 3.1455478056087322460229565127 0.0039551520189390075603131294
19.558985985175549046149591726

```

```

16 3.1418293941968775605768925845 0.000236740607084322114249201
16.706690363137677123496624210

```

```

32 3.1416072961737115420329455901
0.000014642583918303570302206785408 16.167952897192609000701607

```

```

64 3.1415935663851366957920698700 9.127953434573294264867306992
16.041475258673982544799566292

```

```

128 3.1415927106026675271700826829 5.70128742887074392995779141
16.010337223747498929730798518

```

```

256 3.1415926571525228705880812490 3.56272963212543786568431532
16.002582338726316666392092332

```

```

512 3.1415926538124548579967718028
2.2266161953412841952620006776e-10 16.0006454618433291538712487

```

```

1024 3.1415926536037094493404660613
1.3916210877822678018889920820e-11 16.0001613577851963789595768

```

```

2048 3.1415926535906629994496785878
8.6976098703520454491464243430e-13 16.0000403389665991999025824

```

```

4096 3.1415926535898475984900702584
5.4360027426875102236389052510e-14 16.0000100847116688190771675

```

```

8192 3.1415926535897966359638222068
3.3975011788235294010611143751e-15 16.0000025211760588072851619

```

```

16384 3.1415926535897934508064586948
2.1234381531152897976770827770e-16 16.0000006302941553933317377

```

```

32768 3.1415926535897932517341317095
1.3271488326262028481922963535e-17 16.0000001575811599003963972

```

```

65536 3.1415926535897932392921114016
8.2946801834408551584535433293e-19 16.0000000394911672348637263

```

```
131072 3.1415926535897932385144851344
5.1841751110414958326473355711e-20 16.000000111386318990635378

262144 3.1415926535897932384658834927
3.2401094365123258431944666778e-21 16.000000389547783210753098

524288 3.1415926535897932384628458901
2.0250682952682859732650069701e-22 16.000008102594301108858728

1048576 3.1415926535897932384626560399
1.2656670534539545564811850269e-23 16.000079779429821401280808
```

Wow! Is that a giant improvement or what? With this method, we're getting a better estimate for π with a 2048-gon than we were getting before with a 1048576-gon, and our best estimate is now good to 23 digits. Doubling the number of sides now decreases the error by a factor of 16, not 4.

So isn't it impressive how much we can learn about this limit-like process just by doing calculations with a little assistance and paying attention to the results we get? I love this stuff!