



# Survey of Scientific Computing (SciComp 301)

Dave Biersach  
Brookhaven National  
Laboratory  
[dbiersach@bnl.gov](mailto:dbiersach@bnl.gov)



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace SimpleEvents
11 {
12     public partial class Form1 : Form
13     {
14         Person person = new Person();
15
16         public Form1()
17         {
18             InitializeComponent();
19             person.FirstName = "Christian";
20             person.LastName = "Pano";
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             person.MainColor = textBox1.Text;
26         }
27     }
28 }
```

**Session 12**  
Continued Fractions,  
Chi Squared

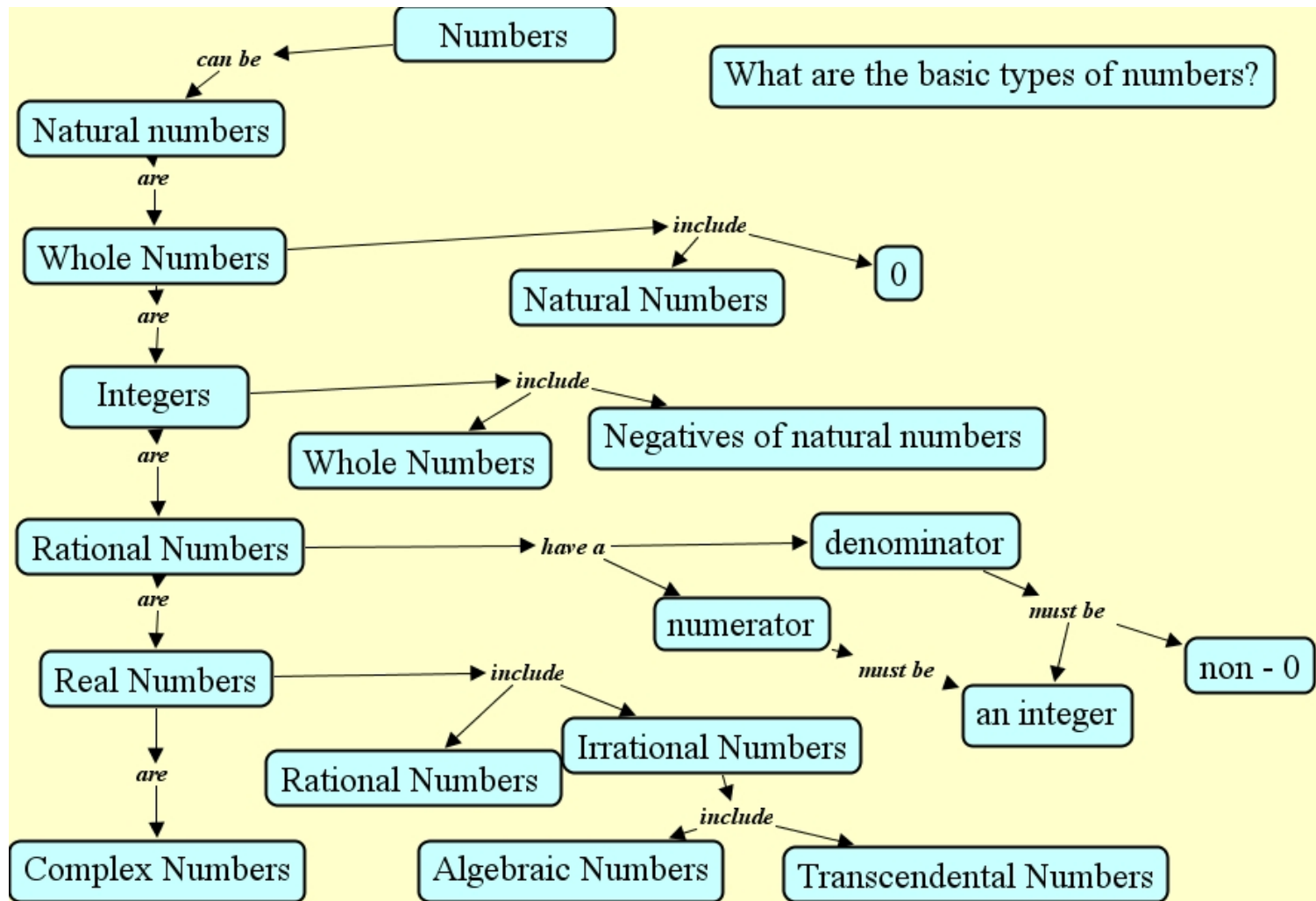
# Session Goals

- Gain an appreciation for **Continued Fractions** in nature
- Understand the three types of CFs: 1) finite, 2) infinite with repeating sequence, 3) infinite with repeating pattern
- Write code to **generate** a generalized CF for a real number, and how to **expand** that CF to produce **convergents** of the original number
- Appreciate the hidden underlying simplicity of the **generalized continued fraction** for  $\pi$
- Perform a computational mathematical experiment to determine the solutions to **Pell's Equation**

# Session Goals

- Gain an appreciation for the **Normal Distribution**
- Investigate if a Normal Distribution can be made from a Uniform Distribution using a **Pachinko game**
- Use **chi-squared statistic** to determine if a random sample conforms to a reasonable Normal Distribution

# Expanding Your Definition of a “Number”



# Continued Fractions

$$x = a_0 + \frac{b_0}{a_1 + \frac{b_1}{a_2 + \frac{b_2}{a_3 + \ddots}}}$$

In a simple continued fraction, all  $b_n = 1$

$$3.245 = \boxed{3} + \frac{1}{\boxed{4} + \frac{1}{\boxed{12} + \frac{1}{\boxed{4}}}}$$

# Continued Fractions

What is the simple CF encoding for **3.245**?

x	floor(x)	x = x - floor(x)	1/x
3.24500000	3	0.24500000	4.08163265
4.08163265	4	0.08163265	12.25000000
12.25000000	12	0.25000000	4.00000000
4.00000000	4	0.00000000	

We stop when this difference is zero

A CF is an *encoding* scheme

**3.245 = [3; 4, 12, 4]**

All **rational** numbers have a CF of **finite** length!

# Continued Fractions

What is the simple CF encoding for **0.825** (= **33/40**)?

x	floor(x)	x = x - floor(x)	1/x
0.82500000	0	0.82500000	1.21212121
1.21212121	1	0.21212121	4.71428571
4.71428571	4	0.71428571	1.40000000
1.40000000	1	0.40000000	2.50000000
2.50000000	2	0.50000000	2.00000000
2.00000000	2	0.00000000	

A CF is an *encoding* scheme

$$0.825 = [0; 1, 4, 1, 2, 2]$$

All **rational** numbers have a CF of **finite** length!

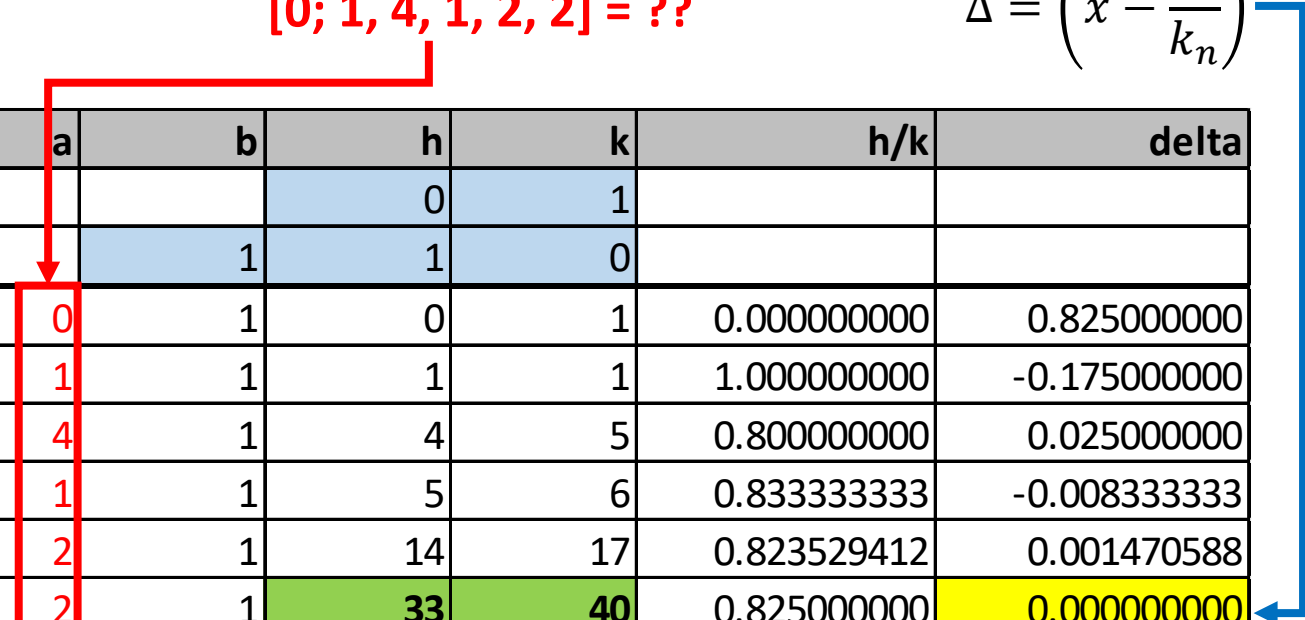
$$\begin{aligned} h_n &= a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)} \\ k_n &= a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)} \end{aligned}$$

# Continued Fractions

How do we expand (decode) a simple CF?

**[0; 1, 4, 1, 2, 2] = ??**

$$\Delta = \left( x - \frac{h_n}{k_n} \right)$$



n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	0	1	0	1	0.000000000	0.825000000
1	1	1	1	1	1.000000000	-0.175000000
2	4	1	4	5	0.800000000	0.025000000
3	1	1	5	6	0.833333333	-0.008333333
4	2	1	14	17	0.823529412	0.001470588
5	2	1	33	40	0.825000000	0.000000000

Each row gives a better and better approximation (**h / k**) to the original number **x**

**[0; 1, 4, 1, 2, 2] = 0.825 (= 33/40)**



# $\sqrt{2}$ to 3,600 digits

1.4142135623730950488016887242096980785696718753769480731766797379907324784621070388503875343276415727350138  
462309122970249248360558507372126441214970999358314132226659275055927557999505011527820605714701095599716059  
702745345968620147285174186408891986095523292304843087143214508397626036279952514079896872533965463318088296  
406206152583523950547457502877599617298355752203375318570113543746034084988471603868999706990048150305440277  
903164542478230684929369186215805784631115966687130130156185689872372352885092648612494977154218334204285686  
060146824720771435854874155657069677653720226485447015858801620758474922657226002085584466521458398893944370  
926591800311388246468157082630100594858704003186480342194897278290641045072636881313739855256117322040245091  
227700226941127573627280495738108967504018369868368450725799364729060762996941380475654823728997180326802474  
420629269124859052181004459842150591120249441341728531478105803603371077309182869314710171111683916581726889  
419758716582152128229518488472089694633862891562882765952635140542267653239694617511291602408715510135150455  
381287560052631468017127402653969470240300517495318862925631385188163478001569369176881852378684052287837629  
389214300655869568685964595155501644724509836896036887323114389415576651040883914292338113206052433629485317  
049915771756228549741438999188021762430965206564211827316726257539594717255934637238632261482742622208671155  
83959992652117625269891754098815934864008345708518147223181420407042650905653233398436457865796796519267292  
39987536661721598257886026336361782749599421940377753681426217738799194551397231274066898329989895386728822  
856378697749662519966583525776198939322845344735694794962952168891485492538904755828834526096524096542889394  
538646625744927556381964410316979833061852019379384940057156333720548068540575867999670121372239475821426306  
585132217408832382947287617393647467837431960001592188807347857617252211867490424977366929207311096369721608  
933708661156734585334833295254675851644710757848602463600834449114818587655554286455123314219926311332517970  
608436559704352856410087918500760361009159465670676883605571740076756905096136719401324935605240185999105062  
108163597726431380605467010293569971042425105781749531057255934984451126922780344913506637568747760283162829  
605532422426957534529028838768446429173282770888318087025339852338122749990812371892540726475367850304821591  
801886167108972869229201197599880703818543332536460211082299279293072871780799888099176741774108983060800326  
311816427988231171543638696617029999341616148786860180455055539869131151860103863753250045581860448040750241  
195184305674533683613674597374423988553285179308960373898915173195874134428817842125021916951875593444387396  
189314549999906107587049090260883517636224749757858858368037457931157339802099986622186949922595913276423619  
410592100328026149874566599688874067956167391859572888642473463585886864496822386006983352642799056283165613  
913942557649062065186021647263033362975075697870606606856498160092718709292153132368281356988937097416504474  
590960537472796524477094099241238710614470543986743647338477454819100872886222149589529591187892149179833981  
083788278153065562315810360648675873036014502273208829351341387227684176678436905294286984908384557445794095  
986260742499549168028530773989382960362133539875320509199893607513906444495768456993471276364507163279154701  
597733548638939423257277540038260274785674172580951416307159597849818009443560379390985590168272154034581581  
521004936662953448827107292396602321638238266612626830502572781169451035379371568823365932297823192986064679  
789864092085609558142614363631004615594332550474493975933999125419532300932175304476533964706627611661753518  
754646209676345587386164880198848497479264045065444896910040794211816925796857563784881498986416854994916357  
614484047021033989215342377037233353115645944389703653166721949049351882905806307401346862641672470110653463  
493916407146285

# Continued Fractions

What is the simple CF encoding for  $\sqrt{2}$  ?

x	floor(x)	x = x - floor(x)	1/x
1.41421356	1	0.41421356	2.41421356
2.41421356	2	0.41421356	2.41421356
2.41421356	2	0.41421356	2.41421356
2.41421356	2	0.41421356	2.41421356
2.41421356	2	0.41421356	2.41421356
2.41421356	2	0.41421356	2.41421356

$$\sqrt{2} = [1; \{2\}]$$

Numbers within {} are repeated

All **irrational numbers** yield an infinite CF  
with a repeated ***sequence*** of finite length!

**There is simple order behind the chaos!**

$$\begin{aligned} h_n &= a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)} \\ k_n &= a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)} \end{aligned}$$

# Continued Fractions

$$\Delta = \left( x - \frac{h_n}{k_n} \right)$$

What fraction best approximates  $\sqrt{2}$  ?

n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	1	1	1	1	1.000000000	0.414213562
1	2	1	3	2	1.500000000	-0.085786438
2	2	1	7	5	1.400000000	0.014213562
3	2	1	17	12	1.416666667	-0.002453104
4	2	1	41	29	1.413793103	0.000420459
5	2	1	99	70	1.414285714	-0.000072152
6	2	1	239	169	1.414201183	0.000012379
7	2	1	577	408	1.414215686	-0.000002124
8	2	1	1393	985	1.414213198	0.000000364
9	2	1	3363	2378	1.414213625	-0.000000063
10	2	1	8119	5741	1.414213552	0.000000011

$$\sqrt{2} \approx 8,119 / 5,741$$

# Continued Fractions

What is the simple CF encoding for  $\sqrt{113}$  ?

x	floor(x)	x = x - floor(x)	1/x
10.63014581	10	0.63014581	1.58693429
1.58693429	1	0.58693429	1.70376823
1.70376823	1	0.70376823	1.42092235
1.42092235	1	0.42092235	2.37573512
2.37573512	2	0.37573512	2.66144940
2.66144940	2	0.66144940	1.51183144
1.51183144	1	0.51183144	1.95376823
1.95376823	1	0.95376823	1.04847275
1.04847275	1	0.04847275	20.63014581
20.63014581	20	0.63014581	1.58693430
1.58693430	1	0.58693430	1.70376822
1.70376822	1	0.70376822	1.42092237
1.42092237	1	0.42092237	2.37573499
2.37573499	2	0.37573499	2.66145027
2.66145027	2	0.66145027	1.51182945
1.51182945	1	0.51182945	1.95377581
1.95377581	1	0.95377581	1.04846442
1.04846442	1	0.04846442	20.63369395
20.63369395	20	0.63369395	1.57804883

Period = 9

$$\sqrt{113} = [10; \{1, 1, 1, 2, 2, 1, 1, 1, 20\}]$$

$$\begin{aligned} h_n &= a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)} \\ k_n &= a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)} \end{aligned}$$

# Continued Fractions

$$\Delta = \left( x - \frac{h_n}{k_n} \right)$$

What fraction best approximates  $\sqrt{113}$  ?

n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	10	1	10	1	10.000000000	0.630145813
1	1	1	11	1	11.000000000	-0.369854187
2	1	1	21	2	10.500000000	0.130145813
3	1	1	32	3	10.666666667	-0.036520854
4	2	1	85	8	10.625000000	0.005145813
5	2	1	202	19	10.631578947	-0.001433135
6	1	1	287	27	10.629629630	0.000516183
7	1	1	489	46	10.630434783	-0.000288970
8	1	1	776	73	10.630136986	0.000008826
9	20	1	16009	1506	10.630146082	-0.000000270
10	1	1	16785	1579	10.630145662	0.000000151
11	1	1	32794	3085	10.630145867	-0.000000054
12	1	1	49579	4664	10.630145798	0.000000015
13	2	1	131952	12413	10.630145815	-0.000000002

$$\sqrt{113} \approx 131,952 / 12,413$$

# $e$ to 3,600 digits

2.7182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274274663  
919320030599218174135966290435729003342952605956307381323286279434907632338298807531952510190115738341879307  
021540891499348841675092447614606680822648001684774118537423454424371075390777449920695517027618386062613313  
845830007520449338265602976067371132007093287091274437470472306969772093101416928368190255151086574637721112  
523897844250569536967707854499699679468644549059879316368892300987931277361782154249992295763514822082698951  
936680331825288693984964651058209392398294887933203625094431173012381970684161403970198376793206832823764648  
042953118023287825098194558153017567173613320698112509961818815930416903515988885193458072738667385894228792  
284998920868058257492796104841984443634632449684875602336248270419786232090021609902353043699418491463140934  
317381436405462531520961836908887070167683964243781405927145635490613031072085103837505101157477041718986106  
873969655212671546889570350354021234078498193343210681701210056278802351930332247450158539047304199577770935  
036604169973297250886876966403555707162268447162560798826517871341951246652010305921236677194325278675398558  
944896970964097545918569563802363701621120477427228364896134225164450781824423529486363721417402388934412479  
635743702637552944483379980161254922785092577825620926226483262779333865664816277251640191059004916449982893  
150566047258027786318641551956532442586982946959308019152987211725563475463964479101459040905862984967912874  
068705048958586717479854667757573205681288459205413340539220001137863009455606881667400169842055804033637953  
764520304024322566135278369511778838638744396625322498506549958862342818997077332761717839280349465014345588  
970719425863987727547109629537415211151368350627526023264847287039207643100595841166120545297030236472549296  
669381151373227536450988890313602057248176585118063036442812314965507047510254465011727211555194866850800368  
532281831521960037356252794495158284188294787610852639813955990067376482922443752871846245780361929819713991  
475644882626039033814418232625150974827987779964373089970388867782271383605772978824125611907176639465070633  
045279546618550966661856647097113444740160704626215680717481877844371436988218559670959102596862002353718588  
748569652200050311734392073211390803293634479727355955277349071783793421637012050054513263835440001863239914  
907054797780566978533580489669062951194324730995876552368128590413832411607226029983305353708761389396391779  
574540161372236187893652605381558415871869255386061647798340254351284396129460352913325942794904337299085731  
580290958631382683291477116396337092400316894586360606458459251269946557248391865642097526850823075442545993  
769170419777800853627309417101634349076964237222943523661255725088147792231519747780605696725380171807763603  
462459278778465850656050780844211529697521890874019660906651803516501792504619501366585436632712549639908549  
144200014574760819302212066024330096412704894390397177195180699086998606636583232278709376502260149291011517  
177635944602023249300280401867723910288097866605651183260043688508817157238669842242201024950551881694803221  
002515426494639812873677658927688163598312477886520141174110913601164995076629077943646005851941998560162647  
907615321038727557126992518275687989302761761146162549356495903798045838182323368612016243736569846703785853  
305275833337939907521660692380533698879565137285593883499894707416181550125397064648171946708348197214488898  
790676503795903669672494992545279033729636162658976039498576741397359441023744329709355477982629614591442936  
451428617158587339746791897571211956187385783644758448423555581050025611492391518893099463428413936080383091  
662818811503715284967059741625628236092168075150177725387402564253470879089137291722828611515915683725241630  
772254406337875931059826760944203261924285317018781772960235413060672136046000389661093647095141417185777014  
180606443636815

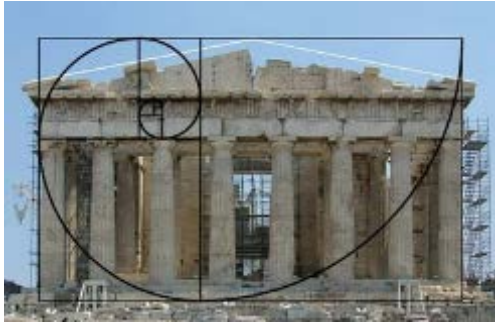
# Continued Fraction for $e$

x	floor(x)	x = x - floor(x)	1/x
2.71828183	2	0.71828183	1.39221119
1.39221119	1	0.39221119	2.54964678
2.54964678	2	0.54964678	1.81935024
1.81935024	1	0.81935024	1.22047929
1.22047929	1	0.22047929	4.53557348
4.53557348	4	0.53557348	1.86715744
1.86715744	1	0.86715744	1.15319313
1.15319313	1	0.15319313	6.52770793
6.52770793	6	0.52770793	1.89498763
1.89498763	1	0.89498763	1.11733388
1.11733388	1	0.11733388	8.52268767
8.52268767	8	0.52268767	1.91318841
1.91318841	1	0.91318841	1.09506427
1.09506427	1	0.09506427	10.51919947
10.51919947	10	0.51919947	1.92604201
1.92604201	1	0.92604201	1.07986461
1.07986461	1	0.07986461	12.52119027
12.52119027	12	0.52119027	1.91868508
1.91868508	1	0.91868508	1.08851229

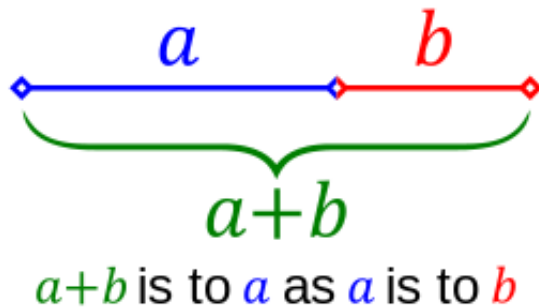
All **transcendental numbers** yield an infinite CF with a repeated **pattern** of finite length

$$e = [2; \{1, 2n, 1\}] \text{ for } n > 0$$

$$e^2 = [7; 2, \{1, 1, 3n, 12n+6, 3n+2\}] \text{ for } n > 0$$



# The Golden Ratio



$$1 + \frac{1}{\varphi} = \varphi$$

$$\varphi + 1 = \varphi^2$$

$$\frac{a+b}{a} = \frac{a}{b} = \varphi$$

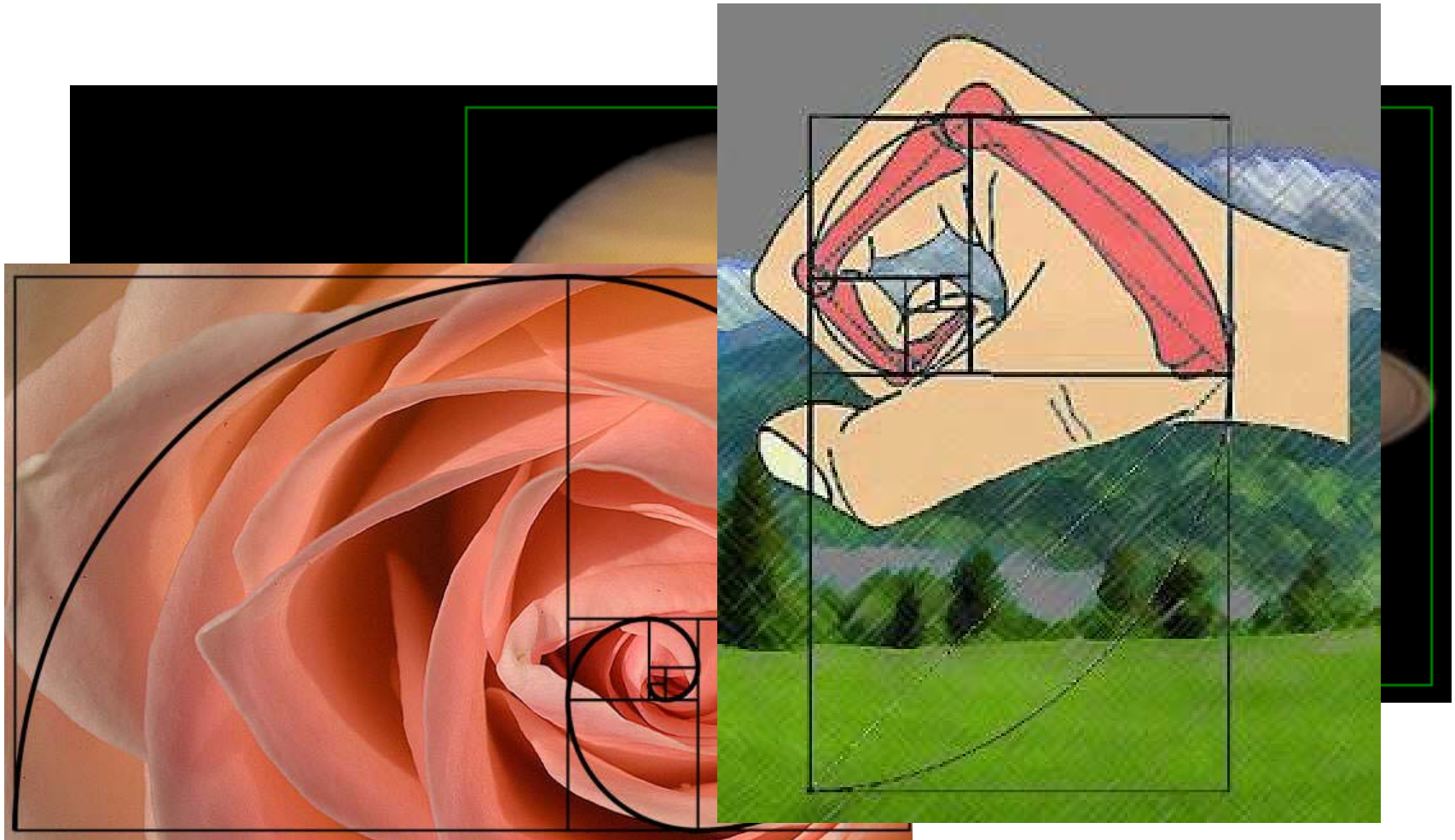
$$\varphi^2 - \varphi - 1 = 0$$

$$1 + \frac{b}{a} = \frac{a}{b} = \varphi$$

$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \dots$$

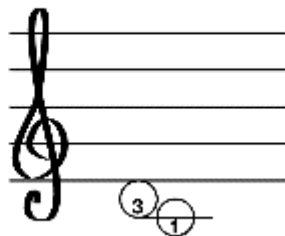
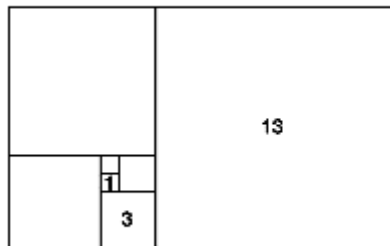


$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \dots$$

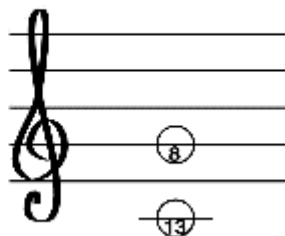
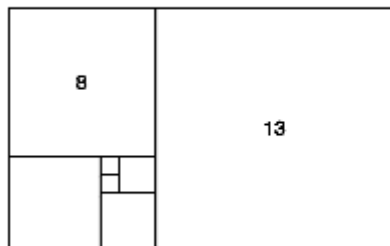


$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \dots$$

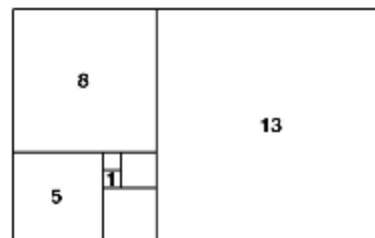
Whole Step



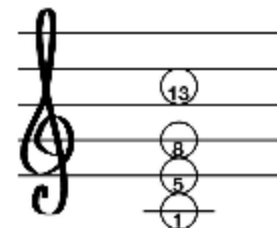
Perfect Fifth



Major Triad

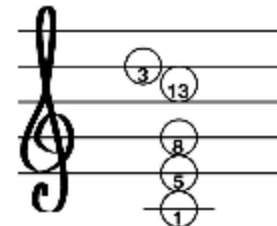


IDEAL PROPORTIONS



IDEAL CHORD STRUCTURE

Major 9



$$\phi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \dots$$

The greatest of luthiers, Stradivarius, designed his violins around the golden ratio ( $\phi$ ). His violins are the most valuable and precious instruments in the string-playing world because of their exquisite tonal and harmonic qualities, [2]. The Stradivarius violin in Fig. 2 reveals how precisely his instruments are determined by the golden ratio, [3]:

$$\frac{a1 + a2}{a2} = \frac{a2}{a1} = \frac{b2}{b1} = \frac{b2}{c2} = \frac{c2}{c1} = \phi$$

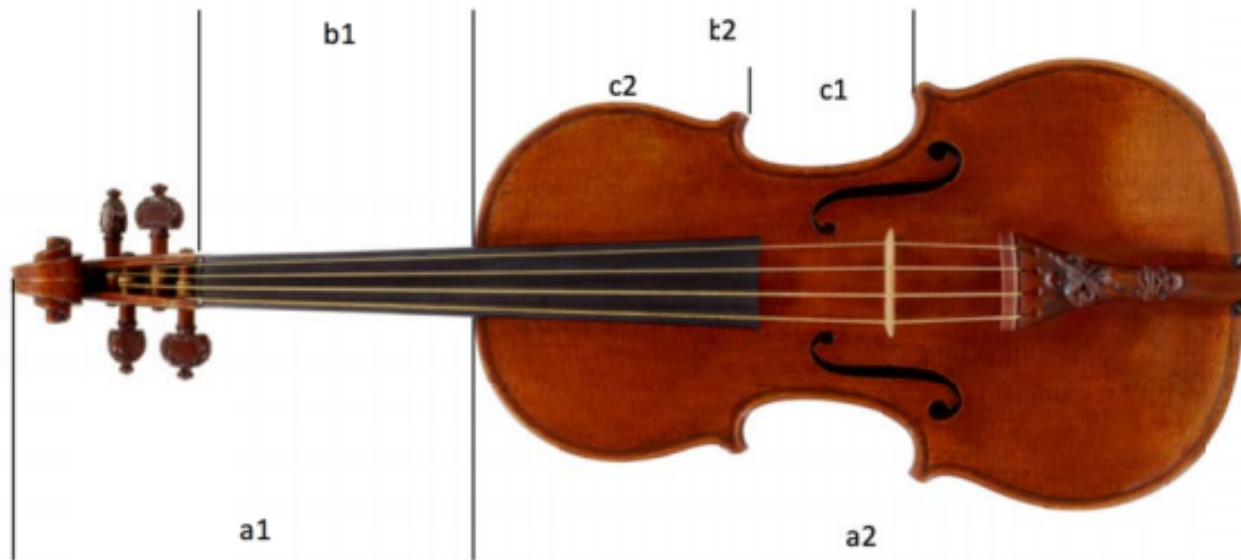


Figure 2. Photo of “Lady Blunt” Stradivarius violin (sold for nearly \$16M).

# $\varphi$ to 3,600 digits

1.6180339887498948482045868343656381177203091798057628621354486227052604628189024497072072041893911374847540  
880753868917521266338622235369317931800607667263544333890865959395829056383226613199282902678806752087668925  
017116962070322210432162695486262963136144381497587012203408058879544547492461856953648644492410443207713449  
470495658467885098743394422125448770664780915884607499887124007652170575179788341662562494075890697040002812  
104276217711177780531531714101170466659914669798731761356006708748071013179523689427521948435305678300228785  
699782977834784587822891109762500302696156170025046433824377648610283831268330372429267526311653392473167111  
211588186385133162038400522216579128667529465490681131715993432359734949850904094762132229810172610705961164  
562990981629055520852479035240602017279974717534277759277862561943208275051312181562855122248093947123414517  
022373580577278616008688382952304592647878017889921990270776903895321968198615143780314997411069260886742962  
267575605231727775203536139362107673893764556060605921658946675955190040055590895022953094231248235521221241  
544400647034056573479766397239494994658457887303962309037503399385621024236902513868041457799569812244574717  
80341731264532204163972321340444948730231541767689375210306873788034417009395440962795589867872320951242689  
355730970450959568440175551988192180206405290551893494759260073485228210108819464454422231889131929468962200  
230144377026992300780308526118075451928877050210968424936271359251876077788466583615023891349333312231053392  
321362431926372891067050339928226526355620902979864247275977256550861548754357482647181414512700060238901620  
777322449943530889990950168032811219432048196438767586331479857191139781539780747615077221175082694586393204  
565209896985556781410696837288405874610337810544439094368358358138113116899385557697548414914453415091295407  
005019477548616307542264172939468036731980586183391832859913039607201445595044977921207612478564591616083705  
949878600697018940988640076443617093341727091914336501371576601148038143062623805143211734815100559013456101  
180079050638142152709308588092875703450507808145458819906336129827981411745339273120809289727922213298064294  
687824274874017450554067787570832373109759151177629784432847479081765180977872684161176325038612112914368343  
767023503711163307258698832587103363222381098090121101989917684149175123313401527338438372345009347860497929  
459915822012581045982309255287212413704361491020547185549611808764265765110605458814756044317847985845397312  
86301625448761148520217064404111660766950597757832570395110878230827106478939021115691039276838453863332156  
582965977310343603232254574363720412440640888267375843395367959312322134373209957498894699565647360072959998  
391288103197426312517971414320123112795518947781726914158911779919564812558001845506563295285985910009086218  
029775637892599916499464281930222935523466747593269516542140210913630181947227078901220872873617073486499981  
562554728113734798716569527489008144384053274837813782466917444229634914708157007352545707089772675469343822  
619546861533120953357923801460927351021011919021836067509730895752895774681422954339438549315533963038072916  
917584610146099505506480367930414723657203986007355076090231731250161320484358364817704848181099160244252327  
167219018933459637860878752870173935930301335901123710239171265904702634940283076687674363865132710628032317  
406931733448234356453185058135310854973335075996677871244905836367541328908624063245639535721252426117027802  
865604323494283730172557440583727826799603173936401328762770124367983114464369476705312724924104716700138247  
831286565064934341803900410178053395058772458665575522939158239708417729833728231152

# Open Lab 1 – Simple CF Encoding

```
int main()
{
    double x = 3.245;
    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms,"
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```

```
vector<int> EncodeCF(double x)
{
    vector<int> terms;
    while(terms.size() < 20)
    {
        terms.push_back(floor(x));
        x = x - floor(x);
        if (x < 1e-9) break;
        x = 1/x;
    }
    return NormalizeCF(terms);
}
```

x	floor(x)	$x = x - \text{floor}(x)$	$1/x$
3.24500000	3	0.24500000	4.08163265
4.08163265	4	0.08163265	12.25000000
12.25000000	12	0.25000000	4.00000000
4.00000000	4	0.00000000	

# View Lab 1 – Simple CF Encoding

```
int main()
{
    double x = 3.245;

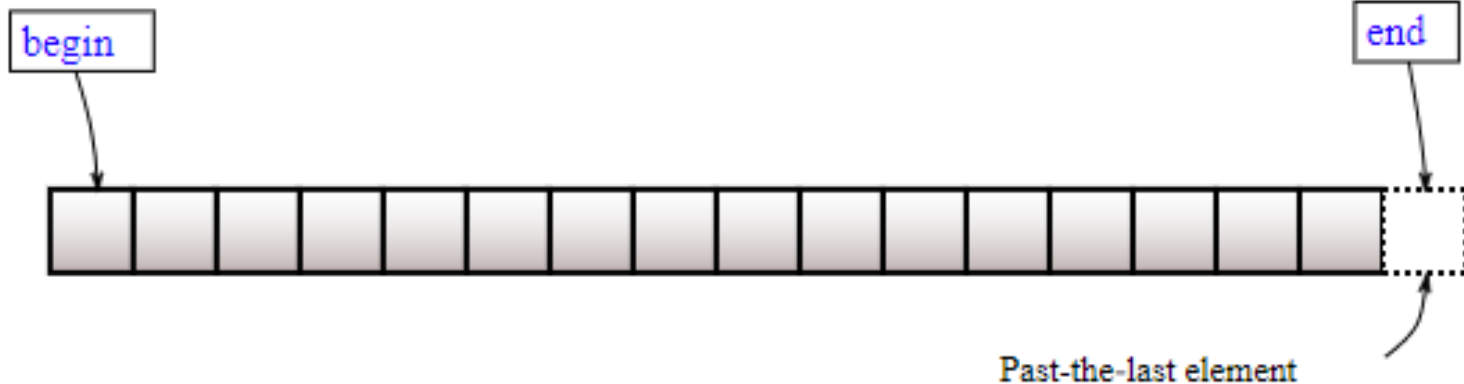
    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms, "
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```

```
void DisplayCF(const vector<int>& terms)
{
    cout << "{";
    auto itr = terms.begin();
    while (true)
    {
        cout << *itr;
        if (++itr == terms.end()) break;
        cout << ", ";
    }
    cout << "}\n";
    return;
}
```



# Run Lab 1 – Simple CF Encoding

```
int main()
{
    double x = 3.245;

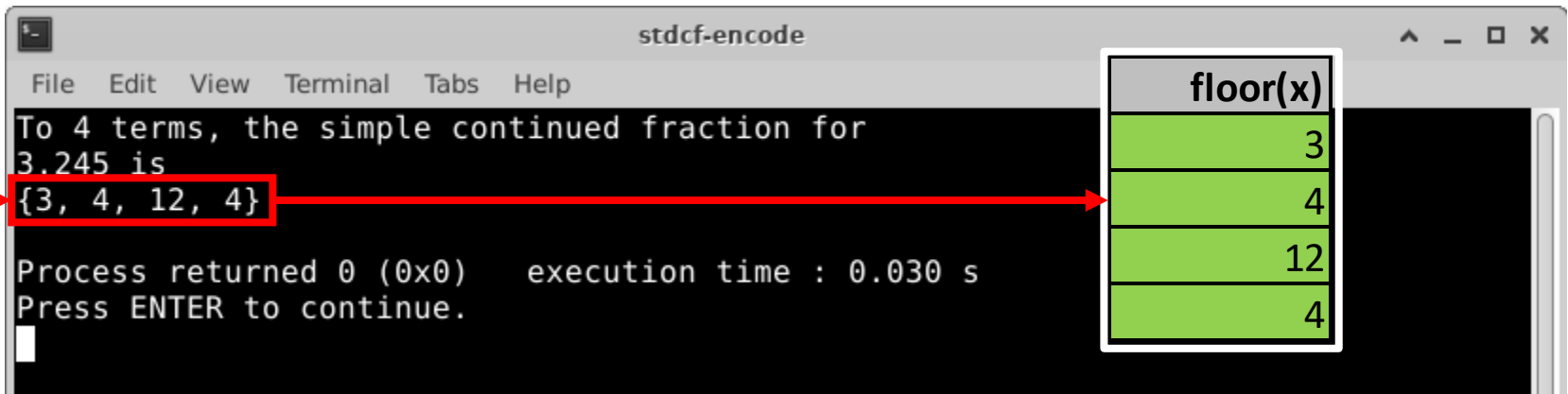
    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms, "
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```

```
void DisplayCF(const vector<int>& terms)
{
    cout << "{";
    auto itr = terms.begin();
    while (true)
    {
        cout << *itr;
        if (++itr == terms.end()) break;
        cout << ", ";
    }
    cout << "}\n";
    return;
}
```



stdcf-encode

File Edit View Terminal Tabs Help

To 4 terms, the simple continued fraction for 3.245 is {3, 4, 12, 4}

Process returned 0 (0x0) execution time : 0.030 s  
Press ENTER to continue.

floor(x)
3
4
12
4

# Edit Lab 1 –Simple CF Encoding

```
int main()
{
    double x = (1 + sqrt(5)) / 2;

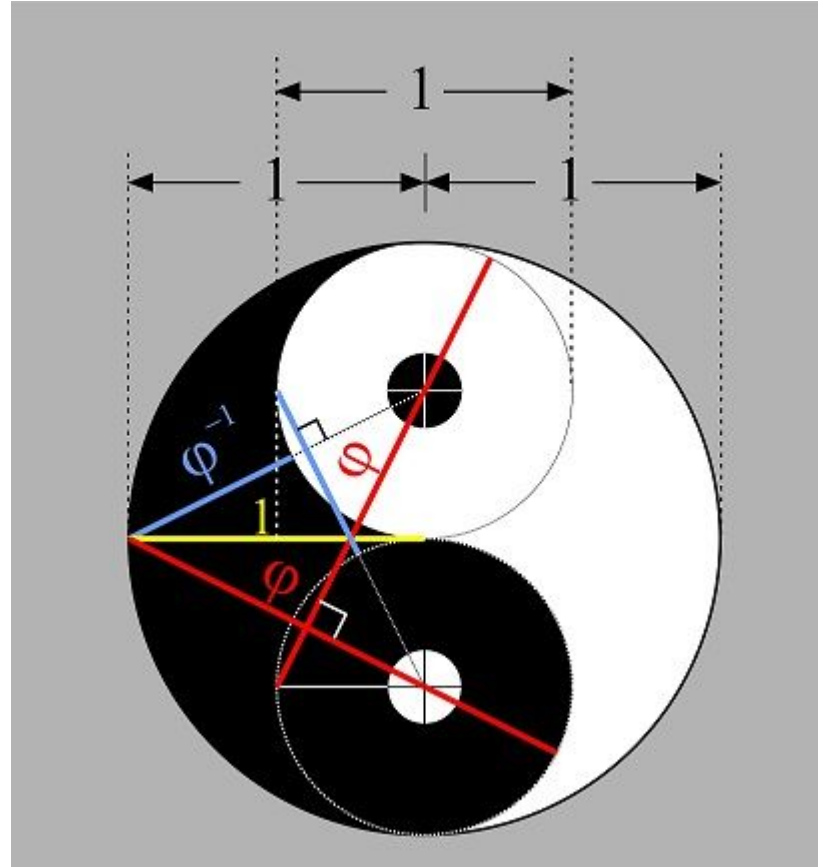
    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms, "
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```

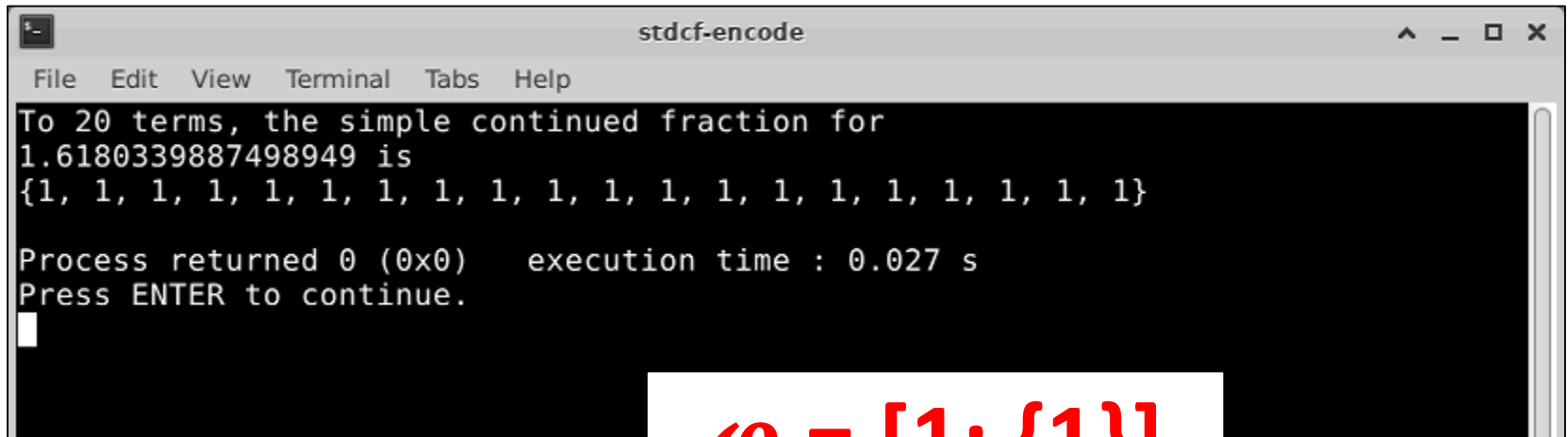
$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887$$





# Run Lab 1 – Simple CF Encoding

- Generate the Simple CF for the golden ratio  $\frac{1+\sqrt{5}}{2}$

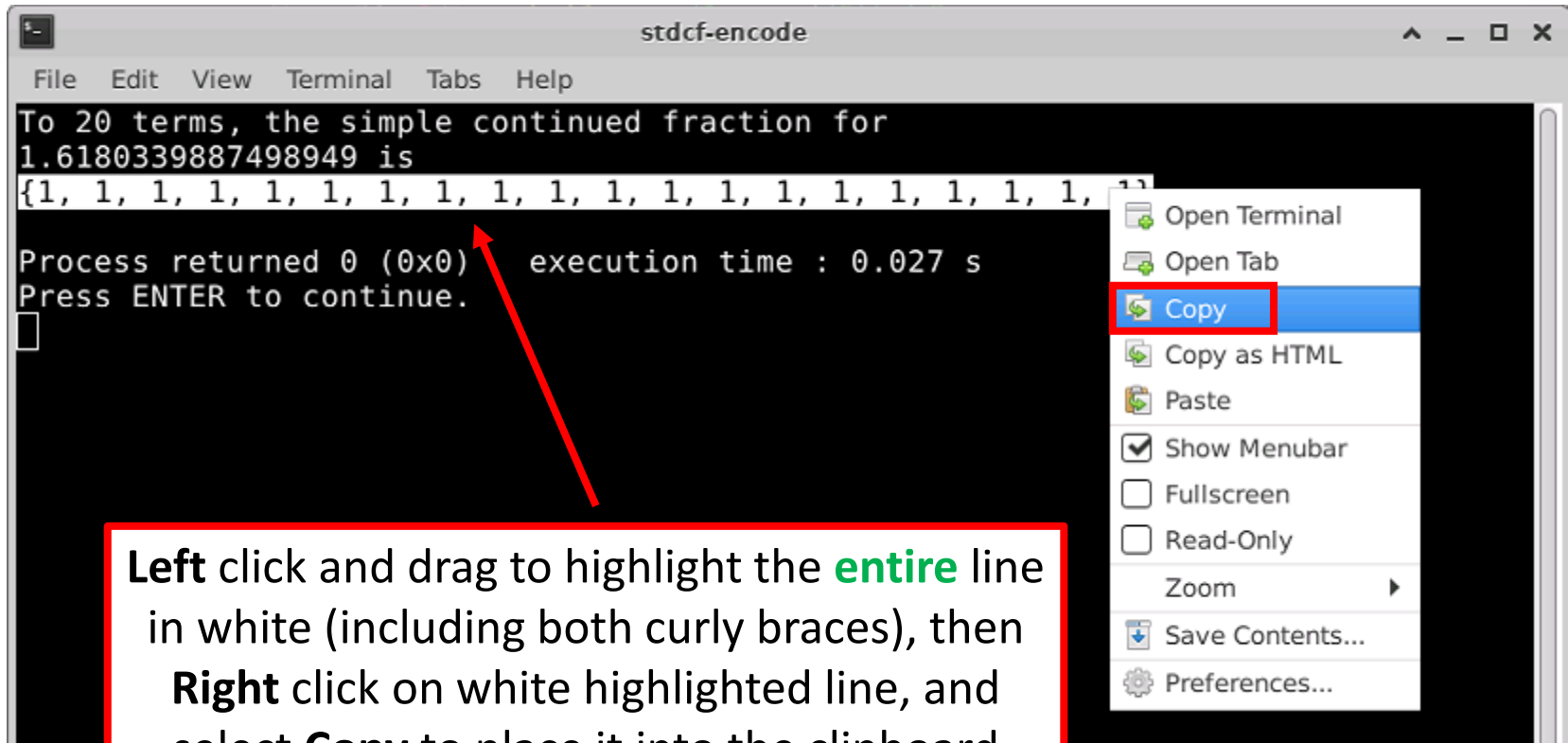


```
stdcf-encode
File Edit View Terminal Tabs Help
To 20 terms, the simple continued fraction for
1.6180339887498949 is
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
Process returned 0 (0x0)   execution time : 0.027 s
Press ENTER to continue.
█
```

$$\varphi = [1; \{1\}]$$

This is Mother  
Nature's true **Unit**  
It is the most simple  
infinite CF possible!

# Check Lab 1 – Simple CF Encoding



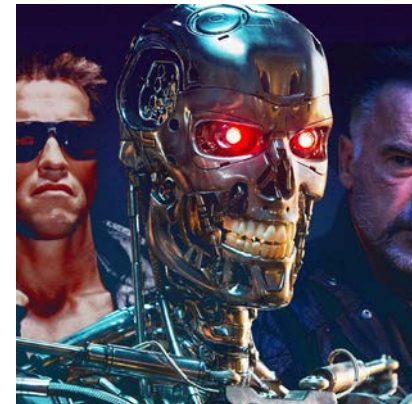
```
stdcf-encode
File Edit View Terminal Tabs Help
To 20 terms, the simple continued fraction for
1.6180339887498949 is
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
Process returned 0 (0x0)   execution time : 0.027 s
Press ENTER to continue.
█
```

Left click and drag to highlight the **entire** line in white (including both curly braces), then **Right** click on white highlighted line, and select **Copy** to place it into the clipboard

# C++ Vector Initialization

```
vector<int> cf{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

- Vectors can be defined using the **list initializer** syntax
  - Elements are comma separated between curly braces
  - First item in list goes into index position **0** in the array
  - The vector is dynamically sized to match the number of elements in the initializer list
- Lab 1 emits source code for Lab 2
- Programs can create programs



# Edit Lab 2

## CF Decode

```
#include "stdafx.h"

using namespace std;

int main()
{
    int maxTerms = 20;

    vector<int> cf(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1);

    vector<double> h(maxTerms + 2);
    vector<double> k(maxTerms + 2);

    if (cf.size() == 0)
    {
        cout << "Error - Missing cf data!";
        return -1;
    }

    h.at(0) = 0;
    k.at(0) = 1;
    h.at(1) = 1;
    k.at(1) = 0;

    cout << "Using " << maxTerms << " terms, ";
    cout << "the continued fraction expansion is:" << endl;
    cout << setw(5) << "a";
    cout << right << setw(15) << "h";
    cout << right << setw(15) << "k";
    cout << setw(20) << "convergent" << endl;

    for (int n{ 2 }; n < maxTerms + 2; ++n)
    {
        double a = cf.at(n - 2);

        h.at(n) = a * h.at(n - 1) + h.at(n - 2);
        k.at(n) = a * k.at(n - 1) + k.at(n - 2);

        double convergent = h.at(n) / k.at(n);

        cout << setprecision(0) << right
              << setw(5) << a << setw(15) << h[n] << setw(15) << k[n]
              << setprecision(14) << fixed << setw(20) << convergent << endl;
    }

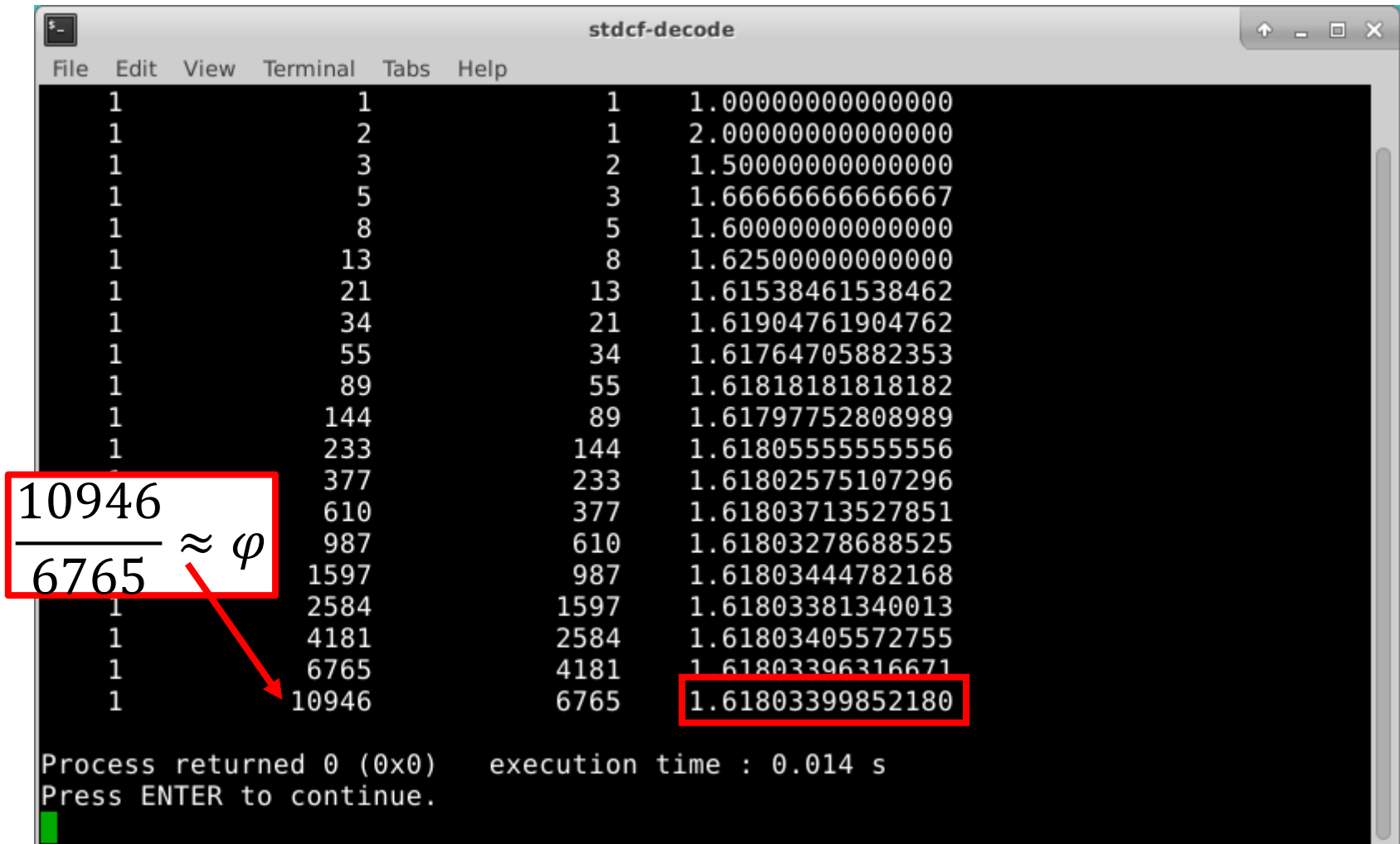
    return 0;
}
```

Don't forget the  
semicolon!

Right click and **paste**  
in the white line  
output from Lab 1

$$\begin{aligned}h_n &= a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)} \\k_n &= a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)}\end{aligned}$$

## Run Lab 2 – Simple CF Decoding



```
stdcf-decode
File Edit View Terminal Tabs Help

1      1      1      1.000000000000000
1      2      1      2.000000000000000
1      3      2      1.500000000000000
1      5      3      1.666666666666667
1      8      5      1.600000000000000
1     13      8      1.625000000000000
1     21     13      1.61538461538462
1     34     21      1.61904761904762
1     55     34      1.61764705882353
1     89     55      1.61818181818182
1    144     89      1.61797752808989
1    233    144      1.61805555555556
1    377    233      1.61802575107296
10946  610     377    1.61803713527851
6765   987     610    1.61803278688525
1597  1597    987    1.61803444782168
2584  2584   1597    1.61803381340013
4181  4181   2584    1.61803405572755
6765  6765    4181    1.61803396316671
10946 10946   6765    1.61803399852180

Process returned 0 (0x0)    execution time : 0.014 s
Press ENTER to continue.
```

$$\frac{10946}{6765} \approx \varphi$$

# $\pi$ to 3,600 digits

3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679821480  
865132823066470938446095505822317253594081284811174502841027019385211055596446229489549303819644288109756659  
334461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155881748  
815209209628292540917153643678925903600113305305488204665213841469519415116094330572703657595919530921861173  
819326117931051185480744623799627495673518857527248912279381830119491298336733624406566430860213949463952247  
371907021798609437027705392171762931767523846748184676694051320005681271452635608277857713427577896091736371  
787214684409012249534301465495853710507922796892589235420199561121290219608640344181598136297747713099605187  
072113499999983729780499510597317328160963185950244594553469083026425223082533446850352619311881710100031378  
387528865875332083814206171776691473035982534904287554687311595628638823537875937519577818577805321712268066  
130019278766111959092164201989380952572010654858632788659361533818279682303019520353018529689957736225994138  
912497217752834791315155748572424541506959508295331168617278558890750983817546374649393192550604009277016711  
390098488240128583616035637076601047101819429555961989467678374494482553797747268471040475346462080466842590  
694912933136770289891521047521620569660240580381501935112533824300355876402474964732639141992726042699227967  
823547816360093417216412199245863150302861829745557067498385054945885869269956909272107975093029553211653449  
872027559602364806654991198818347977535663698074265425278625518184175746728909777727938000816470600161452491  
921732172147723501414419735685481613611573525521334757418494684385233239073941433345477624168625189835694855  
620992192221842725502542568876717904946016534668049886272327917860857843838279679766814541009538837863609506  
800642251252051173929848960841284886269456042419652850222106611863067442786220391949450471237137869609563643  
719172874677646575739624138908658326459958133904780275900994657640789512694683983525957098258226205224894077  
267194782684826014769909026401363944374553050682034962524517493996514314298091906592509372216964615157098583  
874105978859597729754989301617539284681382686838689427741559918559252459539594310499725246808459872736446958  
486538367362226260991246080512438843904512441365497627807977156914359977001296160894416948685558484063534220  
722258284886481584560285060168427394522674676788952521385225499546667278239864565961163548862305774564980355  
936345681743241125150760694794510965960940252288797108931456691368672287489405601015033086179286809208747609  
178249385890097149096759852613655497818931297848216829989487226588048575640142704775551323796414515237462343  
645428584447952658678210511413547357395231134271661021359695362314429524849371871101457654035902799344037420  
073105785390621983874478084784896833214457138687519435064302184531910484810053706146806749192781911979399520  
614196634287544406437451237181921799983910159195618146751426912397489409071864942319615679452080951465502252  
316038819301420937621378559566389377870830390697920773467221825625996615014215030680384477345492026054146659  
252014974428507325186660021324340881907104863317346496514539057962685610055081066587969981635747363840525714  
591028970641401109712062804390397595156771577004203378699360072305587631763594218731251471205329281918261861  
258673215791984148488291644706095752706957220917567116722910981690915280173506712748583222871835209353965725  
121083579151369882091444210067510334671103141267111369908658516398315019701651511685171437657618351556508849  
099898599823873455283316355076479185358932261854896321329330898570642046752590709154814165498594616371802709  
819943099244889575712828905923233260972997120844335732654893823911932597463667305836041428138830320382490375  
898524374417029132765618093773444030707469211201913020330380197621101100449293215160842444859637669838952286  
84783123552658

# Continued Fractions

What is the simple CF encoding for  $\pi$  ?

x	floor(x)	x = x - floor(x)	1/x
3.14159265	3	0.14159265	7.06251331
7.06251331	7	0.06251331	15.99659441
15.99659441	15	0.99659441	1.00341723
1.00341723	1	0.00341723	292.63459088
292.63459088	292	0.63459088	1.57581844
1.57581844	1	0.57581844	1.73665853
1.73665853	1	0.73665853	1.35748105
1.35748105	1	0.35748105	2.79735107
2.79735107	2	0.79735107	1.25415271
1.25415271	1	0.25415271	3.93464232
3.93464232	3	0.93464232	1.06992802
1.06992802	1	0.06992802	14.30041960
14.30041960	14	0.30041960	3.32867763
3.32867763	3	0.32867763	3.04249485
3.04249485	3	0.04249485	23.53226532
23.53226532	23	0.53226532	1.87876228
1.87876228	1	0.87876228	1.13796418
1.13796418	1	0.13796418	7.24825805
7.24825805	7	0.24825805	4.02806683

$\pi = [3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 3, 3, 23, 1, 1, 7, \dots]$   
 (no repeated *pattern* of finite length ☹!)

# Continued Fractions

n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	3	1	3	1	3.000000000	0.141592654
1	7	1	22	7	3.142857143	-0.001264489
2	15	1	333	106	3.141509434	0.000083220
3	1	1	355	113	3.141592920	-0.000000267
4	292	1	103993	33102	3.141592653	0.000000001
5	1	1	104348	33215	3.141592654	0.000000000
6	1	1	208341	66317	3.141592653	0.000000000
7	1	1	312689	99532	3.141592654	0.000000000
8	2	1	833719	265381	3.141592654	0.000000000
9	1	1	1146408	364913	3.141592654	0.000000000
10	3	1	4272943	1360120	3.141592654	0.000000000

If measuring the circumference of Earth:

22 / 7 = accurate to between this classroom and Washington, DC

355 / 113 = accurate to between this classroom and the main parking lot

If measuring the distance between Earth & Sun:

355 / 113 = accurate to 4 football fields

104348 / 33215 = accurate to the length of my shoe

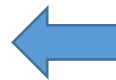


# Generalized Continued Fractions

$$x = a_0 + \frac{b_0}{a_1 + \frac{b_1}{a_2 + \frac{b_2}{a_3 + \ddots}}}$$

In a generalized continued fraction,  
 $a_n$  and  $b_n$  can now be *any* expression

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3}}}$$



In a **simple** CF all the  
numerators ( $b_n$ ) = 1

# Generalized Continued Fractions

What is a **generalized** CF encoding for  $\pi$  ?

$$\pi = 3 + \frac{1^2}{6 + \frac{3^2}{6 + \frac{5^2}{6 + \frac{7^2}{6 + \ddots}}}}$$

Euler



$$\pi = [3; 1, \{(2n+1)^2 | 6\}]$$

$a_0$     $b_0$     $b_n$     $a_n$   
 $\downarrow$     $\downarrow$     $\downarrow$     $\downarrow$

All the mysterious and unpredictable digits of PI  
come from this simple generalized CF !!

# Generalized Continued Fractions

What is a **generalized** CF expansion for  $\pi$  ?

n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	3	1	3	1	3.000000000	0.141592654
1	6	9	19	6	3.166666667	-0.025074013
2	6	25	141	45	3.133333333	0.008259320
3	6	49	1321	420	3.145238095	-0.003645442
4	6	81	14835	4725	3.139682540	0.001910114
5	6	121	196011	62370	3.142712843	-0.0004791
6	6	169	2971101	945945	3.140881341	0.000337
7	6	225	50952465	16216200	3.142071817	-0.000246965
8	6	289	9.74E+08	3.1E+08	3.141254824	
9	6	361	2.06E+10	6.55E+09	3.141839619	

Gen CFs  
converge slowly

$$\pi = [3; 1, \{(2n+1)^2 | 6\}]$$

All the mysterious and unpredictable digits of PI  
come from this simple generalized CF !!

# Generalized Continued Fractions

What is another **generalized** CF encoding for  $\pi$  ?

Biersach



$$\pi = 3 + \frac{1}{7 + \frac{1}{15 + \frac{25}{23 + \frac{65}{31 + \ddots}}}}$$

$$\pi = [3; 1, \{(8n^2-7) | (8n-1)\}]$$

$\begin{matrix} a_0 & b_0 & & b_n & & a_n \\ \downarrow & \downarrow & & \downarrow & & \downarrow \end{matrix}$

There are infinitely many Generalized CFs but not all **converge**

# Generalized Continued Fractions

What is a **generalized** CF expansion for  $\pi$  ?

n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	3	1	3	1	3.000000000	0.141592654
1	7	1	22	7	3.142857143	-0.001264489
2	15	25	333	106	3.141509434	0.000083220
3	23	65	8209	2613	3.141599694	-0.000007040
4	31	121	276124	87893	3.141592618	0.000000035
5	39	193	11762125	3744000	3.141593216	-0.000000562
6	47	281	6.06E+08	1.93E+08	3.141593163	-0.000000510

$$\pi = [3; 1, \{(8n^2-7) | (8n-1)\}]$$

This GCF does  
not converge!

# Generalized Continued Fractions

What is another **generalized** CF encoding for  $\pi$  ?

Biersach



$$\pi = 2 + \frac{8}{6 + \frac{12}{10 + \frac{32}{14 + \frac{60}{18 + \ddots}}}}$$

$$\pi = [2; 8, \{(4n^2+8n) | (4n+2)\}]$$

$a_0$     $b_0$     $b_n$     $a_n$   
 $\downarrow$     $\downarrow$     $\downarrow$     $\downarrow$

# Generalized Continued Fractions

What is a **generalized** CF expansion for  $\pi$  ?

n	a	b	h	k	h/k	delta
-2			0	1		
-1		1	1	0		
0	2	8	2	1	2.000000000	1.141592654
1	6	12	20	6	3.333333333	-0.191740680
2	10	32	224	72	3.111111111	0.030481542
3	14	60	3776	1200	3.146666667	-0.005074013
4	18	96	81408	25920	3.140740741	0.000851913
5	22	140	2153472	685440	3.141736695	-0.000144041
6	26	192	67387392	21450240	3.141568206	0.000024447
7	30	252	2.44E+09	7.75E+08	3.141596814	-0.000004160
8	34	320	9.98E+10	3.18E+10	3.141591945	0.000000709
9	38	396	4.57E+12	1.45E+12	3.141592775	-0.000000121

$$\pi = [2; 8, \{(4n^2+8n) | (4n+2)\}]$$

My GCF for  $\pi$   
converges faster  
than Euler's! 😊

# Generalized Continued Fractions

$$x = a_0 + \frac{b_0}{a_1 + \frac{b_1}{a_2 + \frac{b_2}{a_3 + \ddots}}}$$

In a generalized continued fraction,  
 $a_n$  and  $b_n$  can now be *any* expression

$$\tan(x) = \frac{x}{1 - \frac{x^2}{3 - \frac{x^2}{5 - \frac{x^2}{7 - \ddots}}}}$$

$a_0$   $b_0$   $b_n$   $a_n$   
 $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   
 $\tan(x) = [0; x, \{-x^2 \mid (2n-1)\}]$



# Continued Fractions

- CFs *may* have their own rich **arithmetic**, **algebra**, and potentially even their own **calculus**
  - How can one directly **divide** two CFs?
  - How can one directly take the **sin()** of a CF?
  - What does the **factorial** of a CF look like?
- In many ways a CF is a *more accurate* representation of an irrational or transcendental number
  - The sum of an infinite series must stop somewhere, and after that point, all of the remaining digits of precision are lost
  - A CF encodes the entire number **with no loss of precision**
  - What can you discover about CFs?

# Pell's Equation

- Your scientist has asked you to write a C++ program to find  $x$  &  $y$  for a given  $n$  (assume  $x, y, n \in \mathbb{Z}^+$ ) such that:

$$x^2 - ny^2 = 1$$

- For every  $2 \leq n \leq 70$ , check all  $1 \leq x \leq 70,000$
- Why is there is no need to check for  $y > \left\lfloor \sqrt{\frac{x^2}{n}} \right\rfloor$ ?
- Do you see any relationship between the specific  $x$  &  $y$  values that solve the equation for each  $n$  value?

# Open Lab 3

## Pell's Equation

```
int main()
{
    DisplayHeader();

    const uintmax_t xMax = 70000;

    for (uintmax_t n = 2; n <= 70; n++)
    {
        cout << setw(4) << n;
        bool foundSolution = false;
        uintmax_t x = 1;
        while ((x <= xMax) && !foundSolution)
        {
            uintmax_t xSqr = x * x;
            uintmax_t y = 1;
            uintmax_t yMax = sqrt(xSqr / n);
            while ((y <= yMax) && !foundSolution)
            {
                uintmax_t ySqr = y * y;
                if (xSqr - n * ySqr == 1)
                {
                    cout << setw(8) << x
                        << setw(8) << y;
                    foundSolution = true;
                }
                y++;
            }
            x++;
        }
        if (!foundSolution)
            cout << setw(8) << "- "
                << setw(8) << "- ";
        cout << endl;
    }
    return 0;
}
```

$$x^2 - ny^2 = 1$$

As soon as a valid solution is found for the current value of **n**, then stop trying any more **x** & **y** values

Hyphens indicate no  
solution was found in the  
allowed search space

## Run Lab 3 Pell's Equation

n	x	y
===	=====	=====
2	3	2
3	2	1
4	-	-
5	9	4
6	5	2
7	8	3
8	3	1
9	-	-
10	19	6
11	10	3
12	7	2
13	649	180
14	15	4
15	4	1
16	-	-
17	33	8
18	17	4
19	170	39
20	9	2
21	55	12
22	197	42
23	24	5

24	5	1
25	-	-
26	51	10
27	26	5
28	127	24
29	9801	1820
30	11	2
31	1520	273
32	17	3
33	23	4
34	35	6
35	6	1
36	-	-
37	73	12
38	37	6
39	25	4
40	19	3
41	2049	320
42	13	2
43	3482	531
44	199	30
45	161	24
46	24335	3588
47	48	7

48	7	1
49	-	-
50	99	14
51	50	7
52	649	90
53	66249	9100
54	485	66
55	89	12
56	15	2
57	151	20
58	19603	2574
59	530	69
60	31	4
61	-	-
62	63	8
63	8	1
64	-	-
65	129	16
66	65	8
67	48842	5967
68	33	4
69	7775	936
70	251	30

## Check Lab 3 – Observations

- Which values of  $n$  have no solution?

$$n = 1, 4, 9, 16, 25, 36, 49, \textcolor{red}{61}, 64, \dots$$

- Some of the values for  $x$  &  $y$  are much bigger than for other close values of  $n$ :

40	19	3
41	2049	320
42	13	2
43	3482	531
44	199	30
45	161	24
46	24335	3588
47	48	7
48	7	1

- The *magnitude* of  $n$  does not seem to be a good predictor about the magnitude of the  $x$  &  $y$  values that solve the equation for that specific  $n$

# Pell's Equation: Period of Simple CF

Small values for x & y

n	x	y
35	6	1
47	48	7
60	31	4
68	33	4

$\sqrt{68} =$  Period = 2  
 {8, 4, 16, 4, 16,

Simple CF

Large values for x & y

n	x	y
13	649	180
29	9801	1820
41	2049	320
43	3482	531
46	24335	3588
53	66249	9100
61	1766319049	226153980
67	48842	5967

$\sqrt{29} =$  Period = 5  
 {5, 2, 1, 1, 2, 10, 2, 1, 1, 2, 10,

$\sqrt{53} =$  Period = 5  
 {7, 3, 1, 1, 3, 14, 3, 1, 1, 3, 14,

$\sqrt{61} =$  Period = 11  
 {7, 1, 4, 3, 1, 2, 2, 1, 3, 4, 1, 14, 1, 4, 3, 1, 2, 2, 1,

# Pell's Equation: Period of Simple CF

$$x^2 - ny^2 = 1$$

Large values for x & y		
n	x	y
13	649	180
29	9801	1820
41	2049	320
43	3482	531
46	24335	3588
53	66249	9100
61	1766319049	226153980
67	48842	5967

$\sqrt{13}$

Period = 4

$\sqrt{61}$

Period = 11

Simple CF

If the period of the simple CF of the  $\sqrt{n}$  is large...

...then the x & y will be large for the solution to Pell's equation

# Normal Distribution

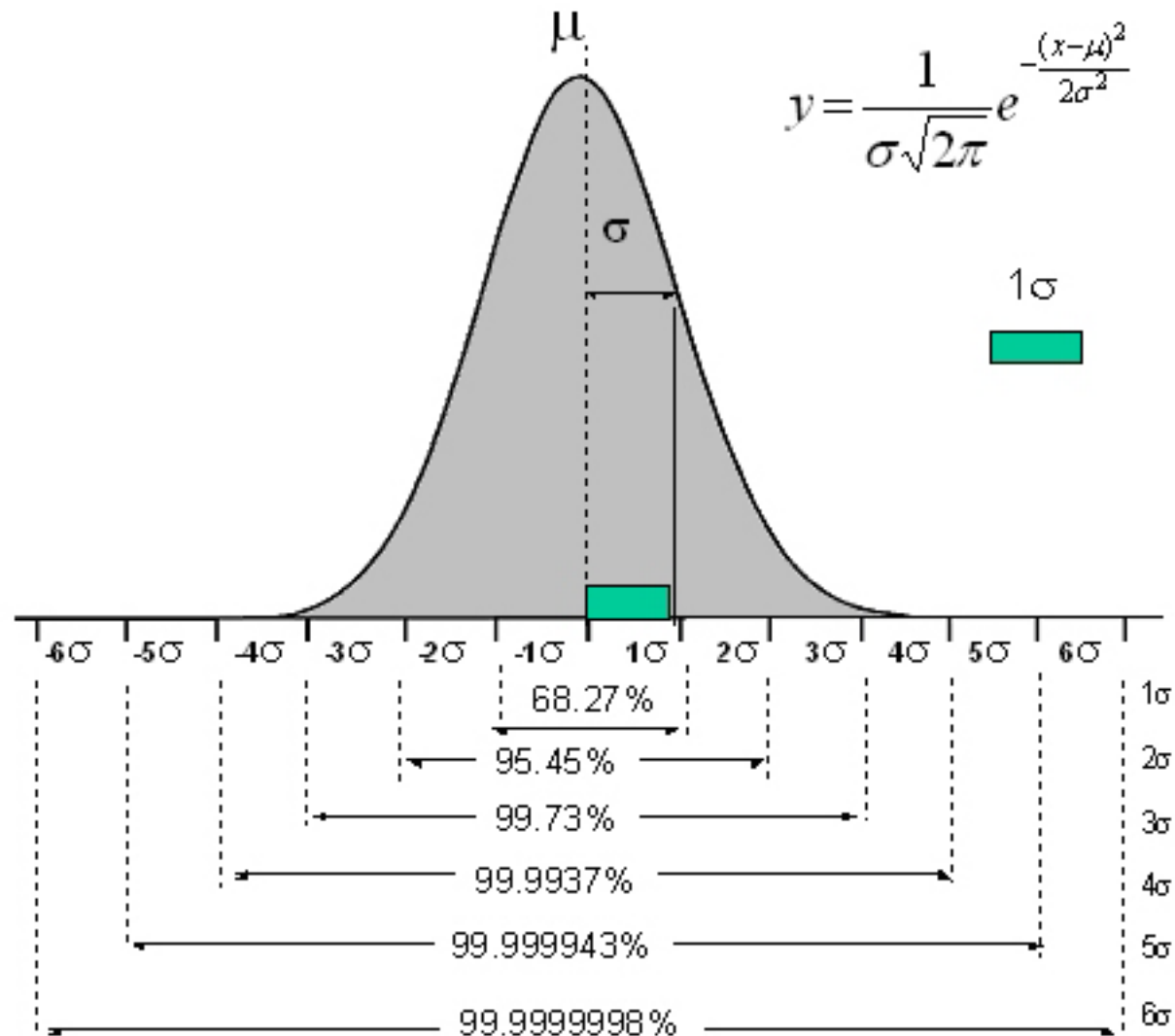
- Until recently, most computer languages only provided a **uniform** pseudo-random number generator
- Growing up I had heard of a **bell curve** and I understood the rationale for **curving** test scores
- However I could not create a **normal distribution** using my 1978 vintage TRS-80 computer using Bill Gate's first **BASIC** language interpreter



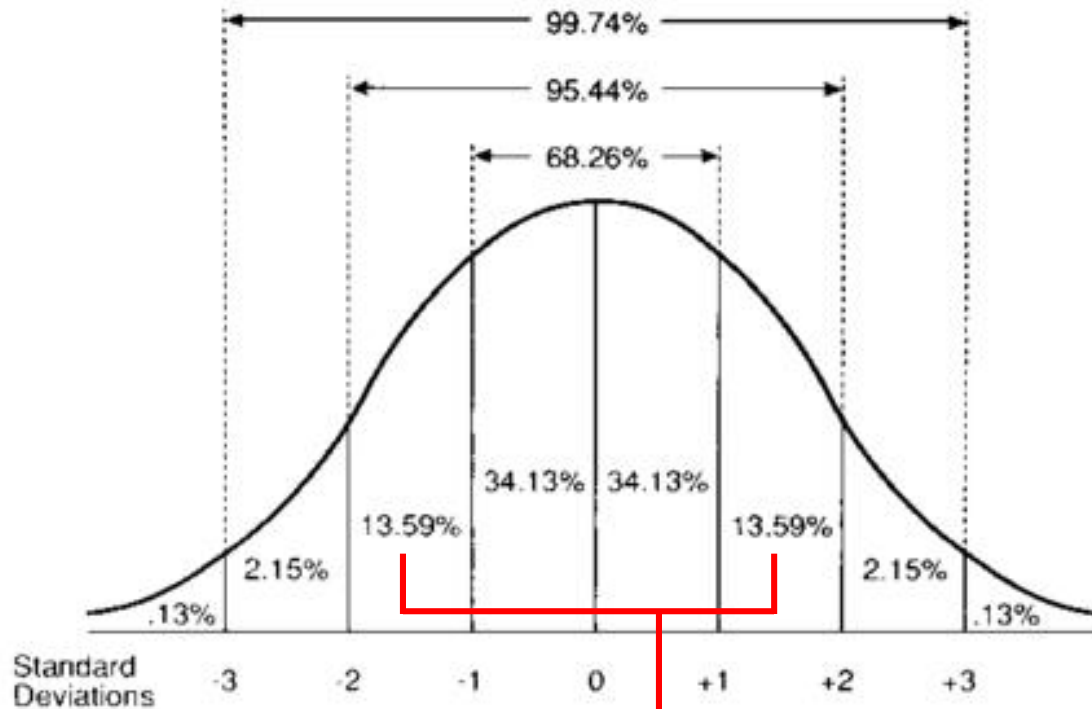
*... or could I?*



# Normal Distribution



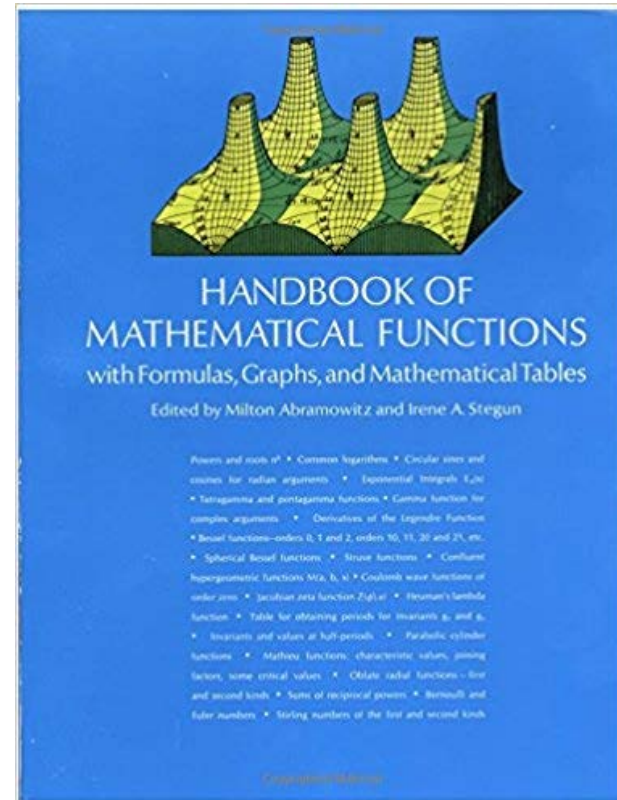
# Normal Distribution



Sigma Number	Population Percent
1	68.26%
2	27.18%
3	4.30%
4	0.26%

# Normal Distribution

- There are indeed several ways to turn a uniform distribution into a normal distribution
- Developing an accurate functional approximation to a normal curve requires advanced mathematics
- Consider this code from Abramowitz & Stegun's classic Handbook

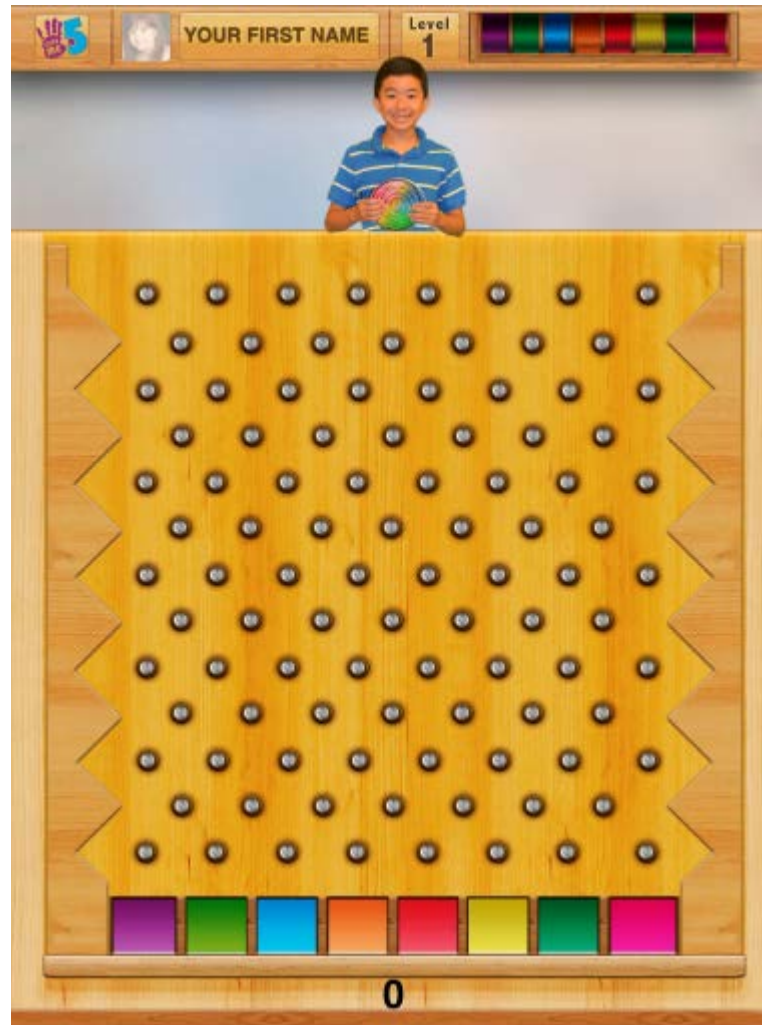


# Normal Distribution

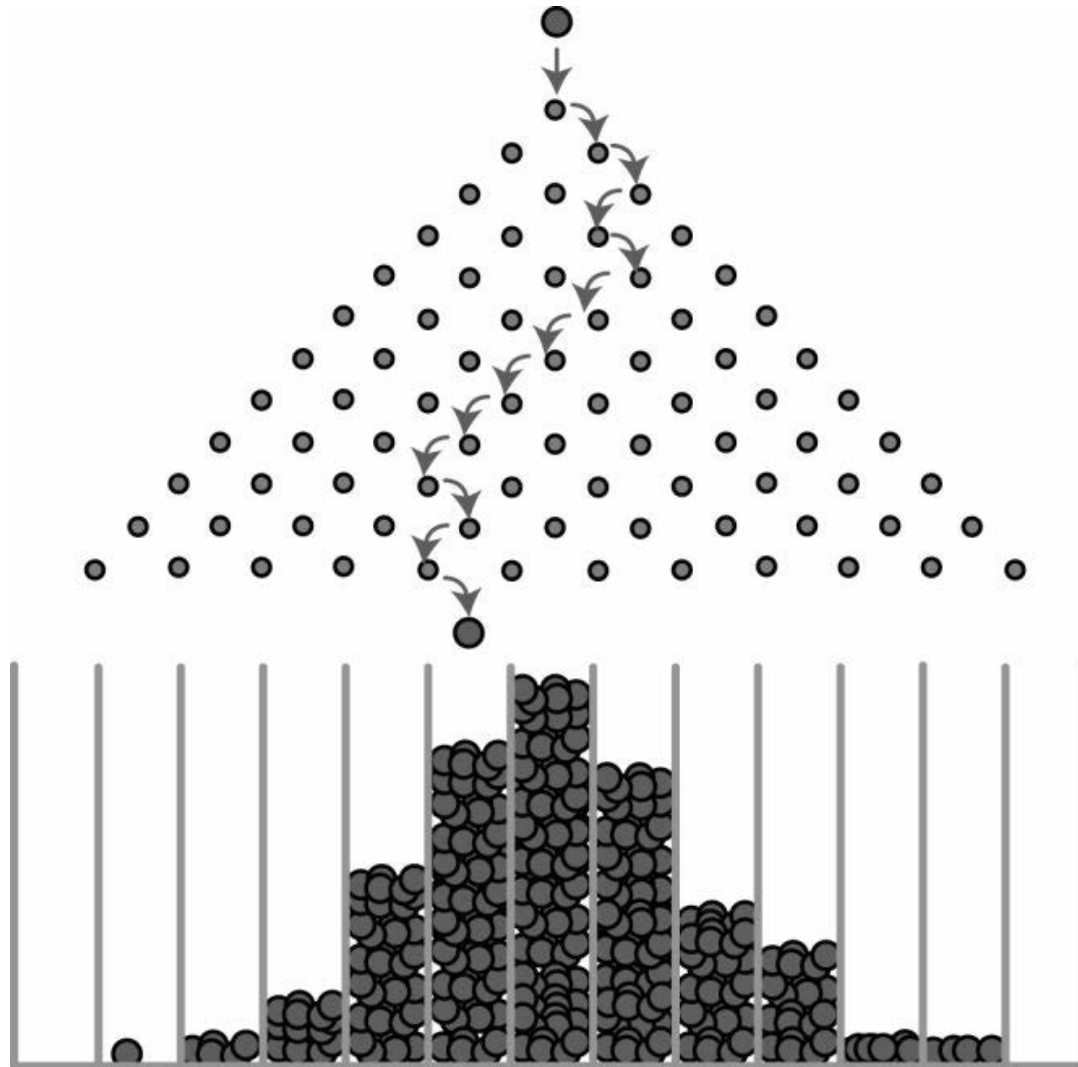
```
double StegunNormal(double mean, double stddev)
{
    double q = 1 - distUniform(generator);
    double p = (q < 0.5) ? q : 1 - q;
    double t = sqrt(log((1 / (p * p)))));
    double x = t - (2.515517 + 0.802853 * t + 0.010328 * (t * t)) /
        (1 + 1.432788 * t + 0.189269 * (t * t) + 0.001308 * (t * t * t));
    x = (q < 0.5) ? x : -1 * x;
    return x * stddev + mean;
}
```

- This was neat but I did not understand it at all!
- Where did all those magic numbers come from?
- I wanted to base my approach after something tangible

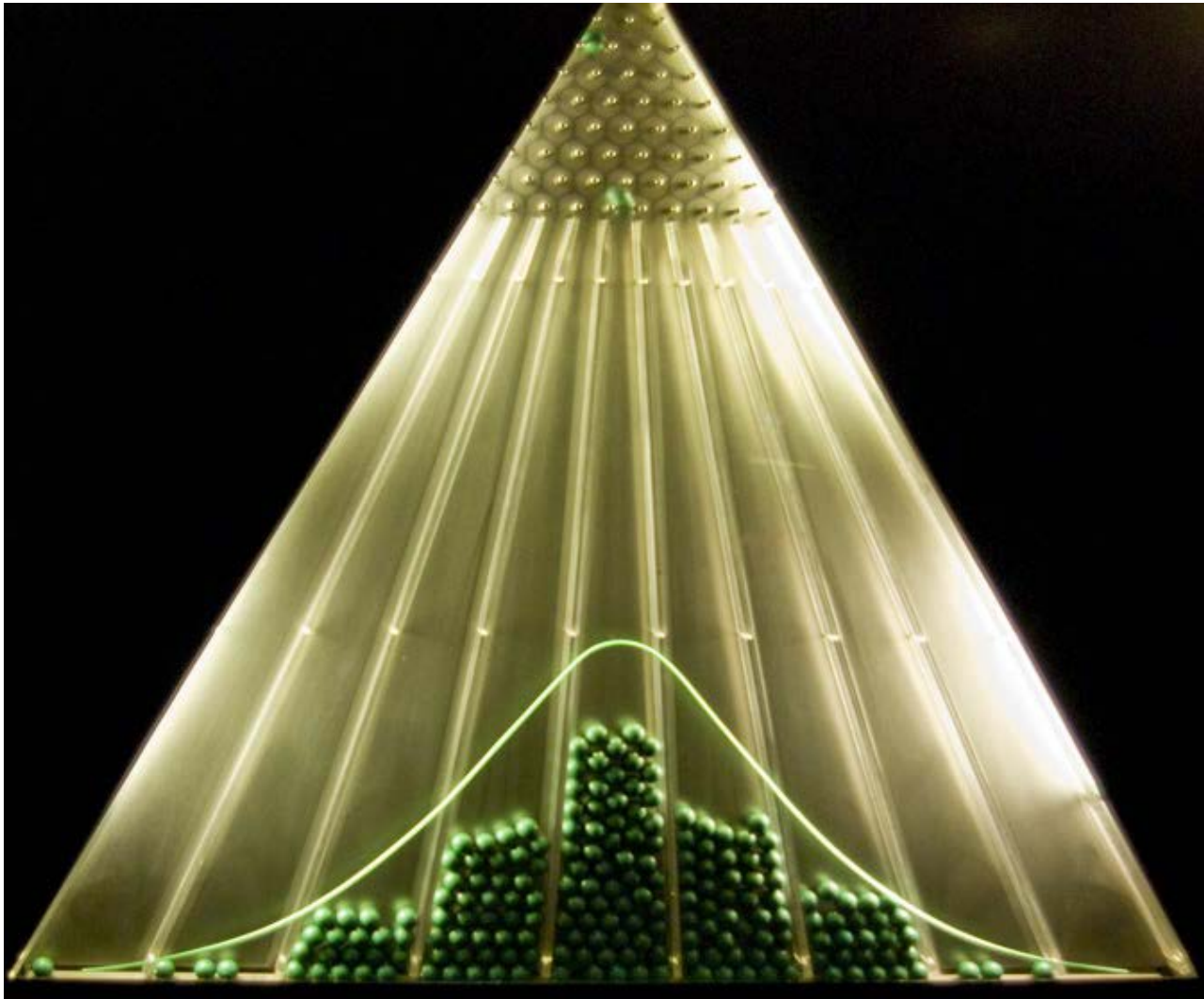
# Pachinko Distribution



# Pachinko Distribution



# Pachinko Distribution

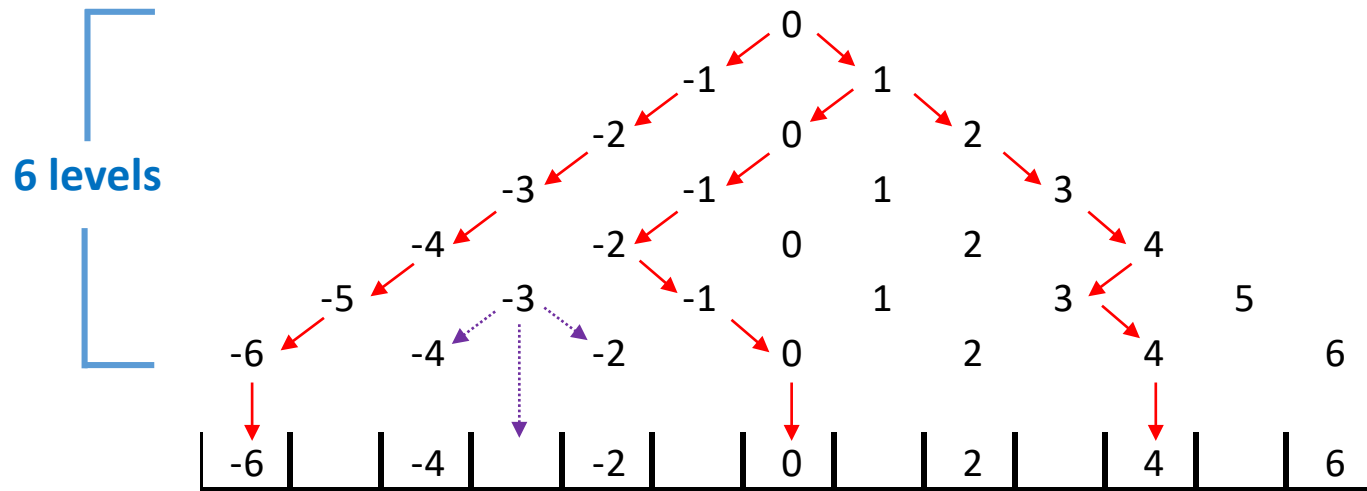


# Normal Distribution

- We can simulate dropping balls down a Pachinko board where at each **level** a ball can move one step to left or right
- If we drop enough balls through enough levels, and we accumulate the **count of balls at each slot** at the bottom, then we should be able to simulate a normal distribution
- We will run a **chi-squared test** to see if our code simulates a distribution that has a reasonable deviation from the perfect (pure) **normal distribution**
- If the discrepancies are *statistically significant*, then we cannot trust that our algorithm is producing a “good enough” normal distribution to use in scientific simulations



# Open Lab 4 – Pachinko Normal



The range is  $\frac{1}{2}$  the number of levels

```
int DropBall()
{
    int slot{};
    for (int level{}; level < levels; level++) {
        int step = distribution(generator);
        if (step == 0)
            slot--;
        else
            slot++;
    }
    slot = slot / 2;
    return slot;
}
```

```
seed_seq seed{ 2016 };
default_random_engine generator(seed);
uniform_int_distribution<int> distribution(0, 1);
```

# View Lab 4 – Pachinko Normal

```
const int balls{ 1000 };
const int levels{ 10 };

seed_seq seed{ 2016 };
default_random_engine generator(seed);
uniform_int_distribution<int> distribution(0, 1);
double mean{};
double stddev{};

const int sigmas{ 4 };
vector<int> sigCountPachinko(sigmas);
vector<int> sigCountNormal(sigmas);

double chiSquared{};
```

```
int main()
{
    CalcMeanPachinko();
    ResetPachinkoDistribution();
    CalcStdDevPachinko();
    ResetPachinkoDistribution();
    CountBallsPerSigma();
    DisplayBallsPerSigma();
    return 0;
}
```

```
void ResetPachinkoDistribution()
{
    generator.seed(seed);
    distribution.reset();
}

void CalcMeanPachinko()
{
    for (int ball{}; ball < balls; ball++)
        mean += DropBall();
    mean /= balls;
}

void CalcStdDevPachinko()
{
    double variance{};
    for (int ball{}; ball < balls; ball++)
        variance += pow(DropBall() - mean, 2);
    stddev = sqrt(variance / balls);
}
```

The mean slot *should be* = 0

# View Lab 4 – Pachinko Normal

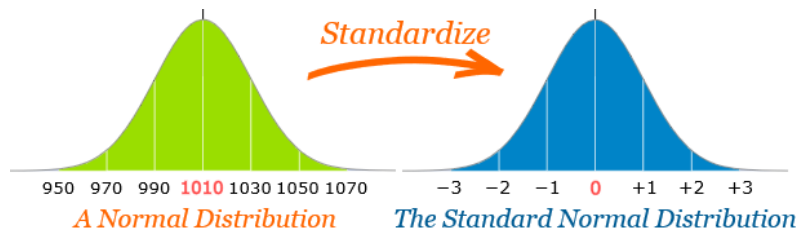
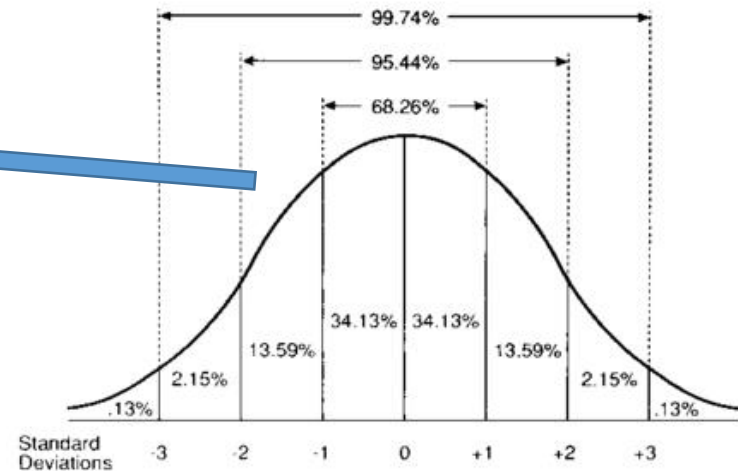
```
const int balls{ 1000 };
const int levels{ 10 };
```

```
const int sigmas{ 4 };
vector<int> sigCountPachinko(sigmas);
vector<int> sigCountNormal(sigmas);
```

```
void CountBallsPerSigma()
{
    sigCountNormal[0] = 0.6826 * balls;
    sigCountNormal[1] = 0.2718 * balls;
    sigCountNormal[2] = 0.0430 * balls;
    sigCountNormal[3] = 0.0026 * balls;

    for (int ball{}; ball < balls; ball++) {
        int slot = DropBall();
        int sigma = abs(mean + slot) / stddev;
        if (sigma < sigCountPachinko.size())
            sigCountPachinko.at(sigma)++;
    }

    for (int s{}; s < sigmas; s++)
        chiSquared += (pow(
            sigCountNormal[s] -
            sigCountPachinko[s], 2)
            / sigCountNormal.at(s));
}
```



- A **sigma** is an integral multiple of the **standard deviation**
- Each **slot** belongs to a sigma
- We count the **number of balls** that fall into each sigma

# Chi-Squared Test

- Does the Pachinko distribution perform reasonably well compared to a perfect **normal** distribution?

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

```
for (int s{}; s < sigmas; s++)  
    chiSquared += (pow(  
        sigCountNormal[s] -  
        sigCountPachinko[s], 2)  
        / sigCountNormal.at(s));
```

Karl Pearson



# Run Lab 4 – Pachinko Normal

```

pachinko-normal
File Edit View Terminal Tabs Help
Balls: 1,000
Levels: 10

Sigma  Pachinko  Normal
1      685      682
2      294      271
3       17       43
4        4        2

Chi-Squared: 19.686
    
```

For 4 degrees of freedom, the **19.686** deviation is statistically **significant** ( $> 9.49$ ) so the proposed algorithm **is not** generating reasonable normally distributed probabilities! ☹☹☹

Degrees of Freedom	Probability										
	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.86	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59
	Nonsignificant								Significant		

# Approaching a Normal Distribution

Maybe we didn't let enough balls drop through to get a good estimate?

```
pachinko-normal
File Edit View Terminal Tabs Help
Balls: 1,000
Levels: 10

Sigma  Pachinko  Normal
1      685      682
2      294      271
3       17      43
4        4        2

Chi-Squared: 19.686

Process returned 0 (0x0)  execution time : 0.018 s
Press ENTER to continue.
```

What if we used 10x more balls?

```
pachinko-normal
File Edit View Terminal Tabs Help
Balls: 10,000
Levels: 10

Sigma  Pachinko  Normal
1     6,570     6,826
2     3,232     2,718
3       180     429
4        18       26

Chi-Squared: 253.789

Process returned 0 (0x0)  execution time : 0.043 s
Press ENTER to continue.
```

Chi-Squared gets even worse!



# Approaching a Normal Distribution

- So can the Pachinko model accurately mimic a normal distribution? **Yes!** ...but only under the right circumstances
- It turns out that **it is not just** the **number of balls** that are used in the experiment that matters, but also the **number of levels** in the simulated Pachinko board
- The **levels** affects how wide (displacement from the center slot) a ball can fall left or right from the topmost (first) pin
- We have to ensure we have enough **levels** (therefore enough **width** at the *bottom* level) to ensure the balls can **spread out** during their fall to occupy the side slots that represent the **higher sigma values**



# Approaching a Normal Distribution

Increasing the **number of levels** improves the agreement of the **sigma ball count** between the Pachinko and perfect normal distribution, **thereby decreasing the chi-squared value**, until the difference **is no longer statistically significant**

Degrees of Freedom	Probability										
	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.86	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59
	Nonsignificant								Significant		

pachinko-normal	pachinko-normal	pachinko-normal																																													
Balls: 1,000 Levels: 10	Balls: 1,000 Levels: 100	Balls: 1,000 Levels: 1,000																																													
<table> <tr><th>Sigma</th><th>Pachinko</th><th>Normal</th></tr> <tr><td>1</td><td>685</td><td>682</td></tr> <tr><td>2</td><td>294</td><td>271</td></tr> <tr><td>3</td><td>17</td><td>43</td></tr> <tr><td>4</td><td>4</td><td>2</td></tr> </table>	Sigma	Pachinko	Normal	1	685	682	2	294	271	3	17	43	4	4	2	<table> <tr><th>Sigma</th><th>Pachinko</th><th>Normal</th></tr> <tr><td>1</td><td>644</td><td>682</td></tr> <tr><td>2</td><td>301</td><td>271</td></tr> <tr><td>3</td><td>52</td><td>43</td></tr> <tr><td>4</td><td>3</td><td>2</td></tr> </table>	Sigma	Pachinko	Normal	1	644	682	2	301	271	3	52	43	4	3	2	<table> <tr><th>Sigma</th><th>Pachinko</th><th>Normal</th></tr> <tr><td>1</td><td>662</td><td>682</td></tr> <tr><td>2</td><td>295</td><td>271</td></tr> <tr><td>3</td><td>41</td><td>43</td></tr> <tr><td>4</td><td>2</td><td>2</td></tr> </table>	Sigma	Pachinko	Normal	1	662	682	2	295	271	3	41	43	4	2	2
Sigma	Pachinko	Normal																																													
1	685	682																																													
2	294	271																																													
3	17	43																																													
4	4	2																																													
Sigma	Pachinko	Normal																																													
1	644	682																																													
2	301	271																																													
3	52	43																																													
4	3	2																																													
Sigma	Pachinko	Normal																																													
1	662	682																																													
2	295	271																																													
3	41	43																																													
4	2	2																																													
Chi-Squared: 19.686	Chi-Squared: 7.822	Chi-Squared: 2.805																																													
Process returned 0 (0x0)    exec Press ENTER to continue.	Process returned 0 (0x0)    exec Press ENTER to continue.	Process returned 0 (0x0)    exec Press ENTER to continue.																																													



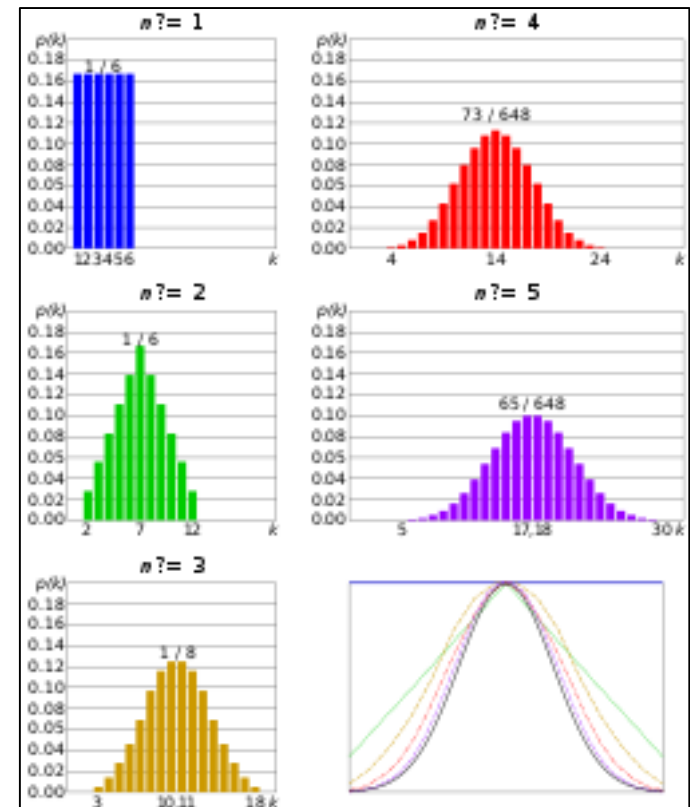
# Central Limit Theorem

## Central limit theorem

In probability theory, the **central limit theorem (CLT)** establishes that, for the most commonly studied scenarios, when independent random variables are added, their sum tends toward a normal distribution

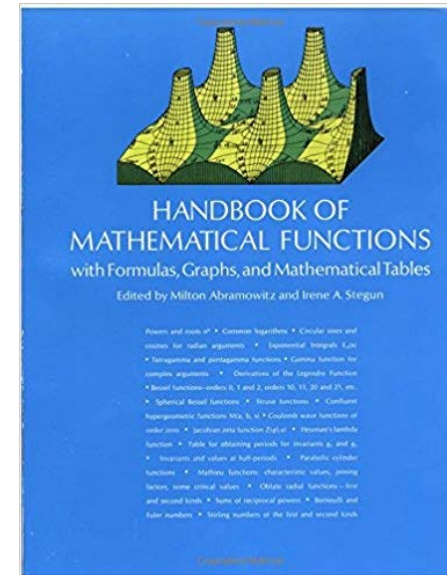
(commonly known as a *bell curve*) even if the original variables themselves are not normally distributed. In more precise terms, given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined (finite) expected value and finite variance, will be approximately normally distributed, regardless of the underlying distribution.<sup>[1][2]</sup> The

theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.



# Approximating a Normal Distribution

It is best to use the code from Abramowitz & Stegun, as only **one** call to this function is required to produce a normalized random variable versus to my method:  
(1000 balls x 1000 levels = **1M** calls!)



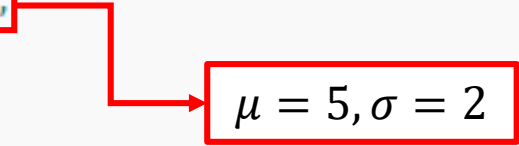
```
double StegunNormal(double mean, double stddev)
{
    double q = 1 - distUniform(generator);
    double p = (q < 0.5) ? q : 1 - q;
    double t = sqrt(log((1 / (p * p)))));
    double x = t - (2.515517 + 0.802853 * t + 0.010328 * (t * t)) /
        (1 + 1.432788 * t + 0.189269 * (t * t) + 0.001308 * (t * t * t));
    x = (q < 0.5) ? x : -1 * x;
    return x * stddev + mean;
}
```

# The C++ has a built-in `normal_distribution()`

```
int main()
{
    std::random_device rd{};
    std::mt19937 gen{rd()};

    // values near the mean are the most likely
    // standard deviation affects the dispersion of generated values from the mean
    std::normal_distribution<> d{5,2};

    std::map<int, int> hist{};
    for(int n=0; n<10000; ++n) {
        ++hist[std::round(d(gen))];
    }
    for(auto p : hist) {
        std::cout << std::setw(2)
                    << p.first << ' ' << std::string(p.second/200, '*') << '\n';
    }
}
```



A red box highlights the code `d{5,2};` in the C++ code. A red arrow points from the `2` in this code to a red box containing the mathematical expression  $\mu = 5, \sigma = 2$ . This indicates that the first parameter (5) is the mean ( $\mu$ ) and the second parameter (2) is the standard deviation ( $\sigma$ ).

# Now you know...

- Rational, Irrational, and Transcendental numbers each have their own style of continued fractions
  - We can take any real number and *generate* a CF
  - Given a CF, we can *expand* it to regain the original number
- The *convergents* of a CF are excellent approximations to the original number
- The magnitude of the **x** & **y** values in solutions to **Pell's Equation**  $\{x^2 - ny^2 = 1\}$  is related to the period of the *simple* continued fraction of  $\sqrt{n}$
- Memorizing thousands of digits of  $\pi$  is okay – but I'd rather appreciate its beautifully simple GCF:  **$[3; 1, \{6 | (2n+1)^2\}]$**

## Now you know...

- A perfect **Normal distribution** ensures that **68.26%** of all values fall within **one** (1) standard deviation from the **mean**
  - 99.73% of all values in a perfect normal distribution are within **three** (3) standard deviations from the mean
  - The normal distribution is known as the “**bell curve**”
- There is a way to convert a PRNG created uniform distribution into a normal distribution – but **don't use the Pachinko method**
- The **chi-squared** test suggests if the discrepancies between the observed and the expected values are statistically significant