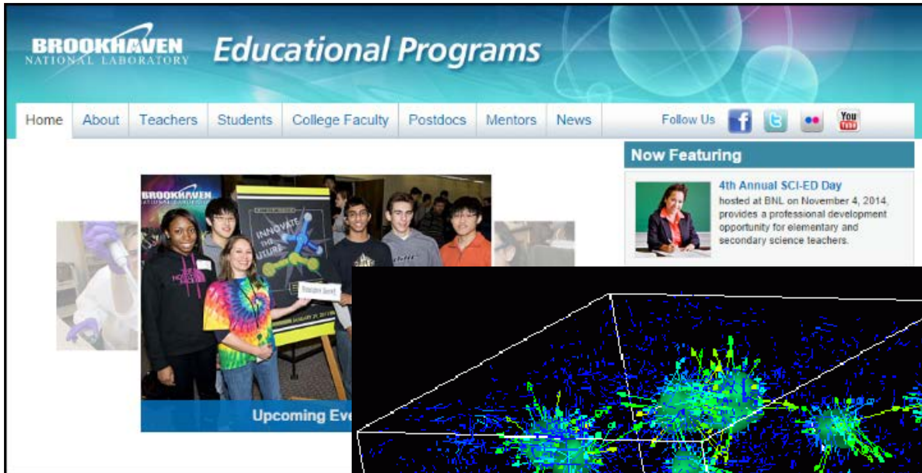
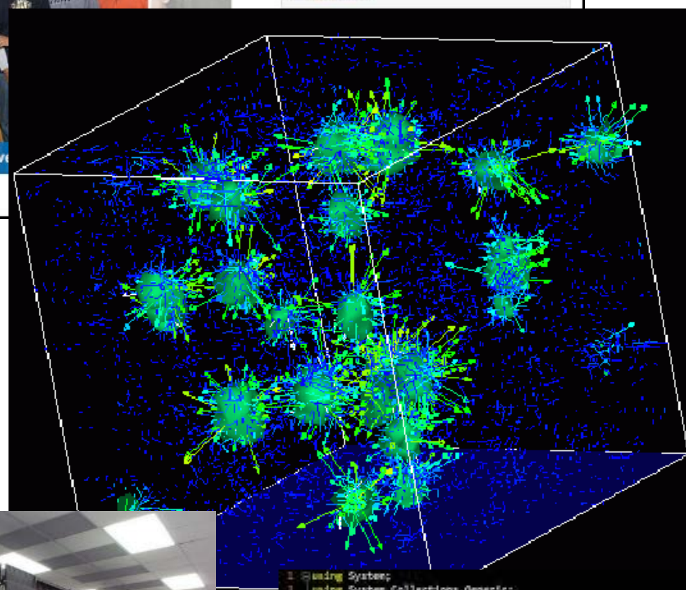
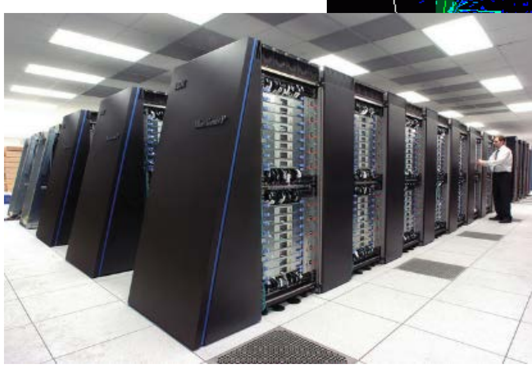






Survey of Scientific Computing

(SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Session 12

Continued Fractions, Chi Squared

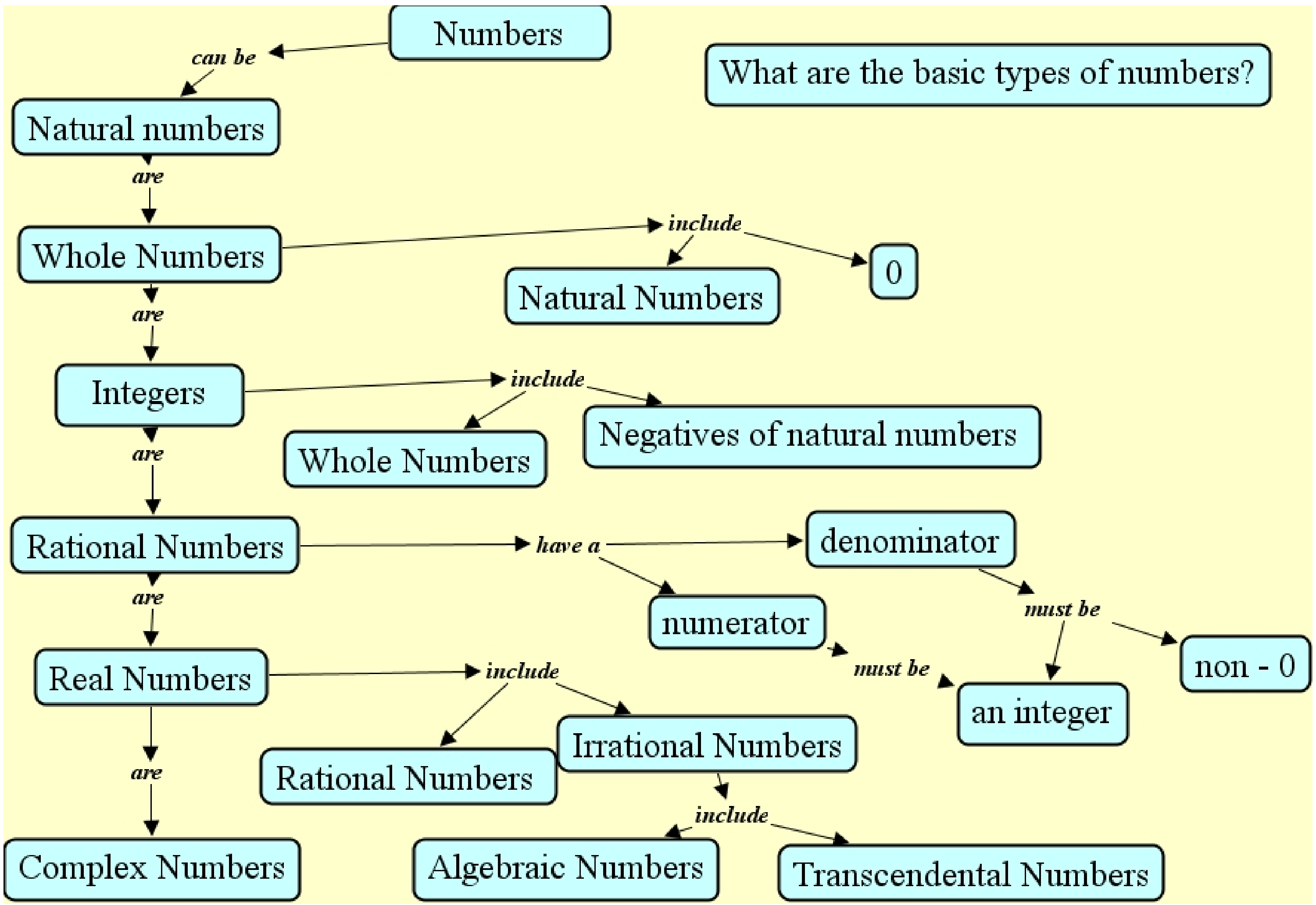
Session Goals

- 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 π
- 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

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

In a <u>simple</u> continued fraction, all $b_n = 1$

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

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!

$$h_n = a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)}$$
$$k_n = a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)}$$

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

1.41421356237309504880168872420969807856967187537694807317667973799073247846210703885038753432764157273501384
62309122970249248360558507372126441214970999358314132226659275055927557999505011527820605714701095599716059
70274534596862014728517418640889198609552329230484308714321450839762603627995251407989687253396546331808829
64062061525835239505474575028775996172983557522033753185701135437460340849884716038689997069900481503054402
77903164542478230684929369186215805784631115966687130130156185689872372352885092648612494977154218334204285
68606014682472077143585487415565706967765372022648544701585880162075847492265722600208558446652145839889394
43709265918003113882464681570826301005948587040031864803421948972782906410450726368813137398552561173220402
45091227700226941127573627280495738108967504018369868368450725799364729060762996941380475654823728997180326
80247442062926912485905218100445984215059112024944134172853147810580360337107730918286931471017111168391658
17268894197587165821521282295184884720896946338628915628827659526351405422676532396946175112916024087155101
35150455381287560052631468017127402653969470240300517495318862925631385188163478001569369176881852378684052
28783762938921430065586956868596459515550164472450983689603688732311438941557665104088391429233811320605243
36294853170499157717562285497414389991880217624309652065642118273167262575395947172559346372386322614827426
22208671155839599926521176252698917540988159348640083457085181472231814204070426509056532333398436457865796
79651926729239987536661721598257886026336361782749599421940377775368142621773879919455139723127406689832998
98953867288228563786977496625199665835257761989393228453447356947949629521688914854925389047558288345260965
24096542889394538646625744927556381964410316979833061852019379384940057156333720548068540575867999670121372
23947582142630658513221740883238294728761739364746783743196000159218880734785761725221186749042497736692920
73110963697216089337086611567345853348332952546758516447107578486024636008344491148185876555542864551233142
19926311332517970608436559704352856410087918500760361009159465670676883605571740076756905096136719401324935
60524018599910506210816359772643138060546701029356997104242510578174953105725593498445112692278034491350663
75687477602831628296055324224269575345290288387684464291732827708883180870253398523381227499908123718925407
26475367850304821591801886167108972869229201197599880703818543332536460211082299279293072871780799888099176
74177410898306080032631181642798823117154363869661702999934161614878686018045505553986913115186010386375325
00455818604480407502411951843056745336836136745973744239885532851793089603738989151731958741344288178421250
21916951875593444387396189314549999906107587049090260883517636224749757858858368037457931157339802099986622
18694992259591327642361941059210032802614987456659968887406795616739185957288864247346358588686449682238600
69833526427990562831656139139425576490620651860216472630333629750756978706066068564981600927187092921531323
68281356988937097416504474590960537472796524477094099241238710614470543986743647338477454819100872886222149
58952959118789214917983398108378827815306556231581036064867587303601450227320882935134138722768417667843690
52942869849083845574457940959862607424995491680285307739893829603621335398753205091998936075139064444957684
56993471276364507163279154701597733548638939423257277540038260274785674172580951416307159597849818009443560
37939098559016827215403458158152100493666295344882710729239660232163823826661262683050257278116945103537937
15688233659322978231929860646797898640920856095581426143636310046155943325504744939759339991254195323009321
75304476533964706627611661753518754646209676345587386164880198848497479264045065444896910040794211816925796
85756378488149898641685499491635761448404702103398921534237703723335311564594438970365316672194904935188290
58063074013468626416724701106534634939164071462

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!

$$h_n = a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)}$$
$$k_n = a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)}$$

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}]

$$h_n = a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)}$$
$$k_n = a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)}$$

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

2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742746 63
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

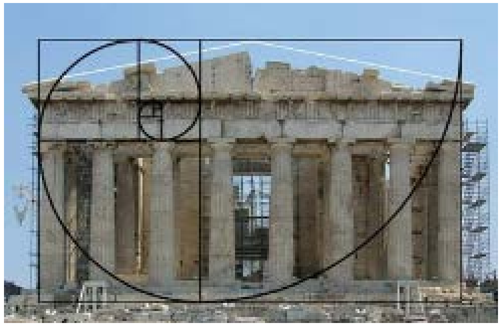
Continued Fraction for *e*

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
[illegible]	1	0.89498763	1.11733388
[illegible]	1	0.11733388	8.52268767
[illegible]	8	0.52268767	1.91318841
[illegible]	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

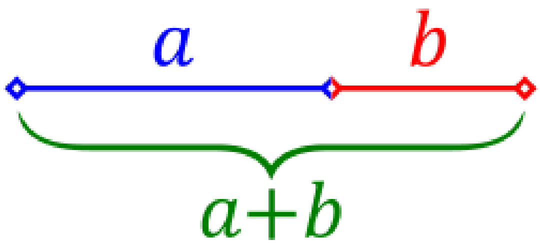
All transcendental numbers yield an infinite CF with a repeated *pattern* of finite length

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

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



The Golden Ratio



$a+b$ is to a as a is to b

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

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

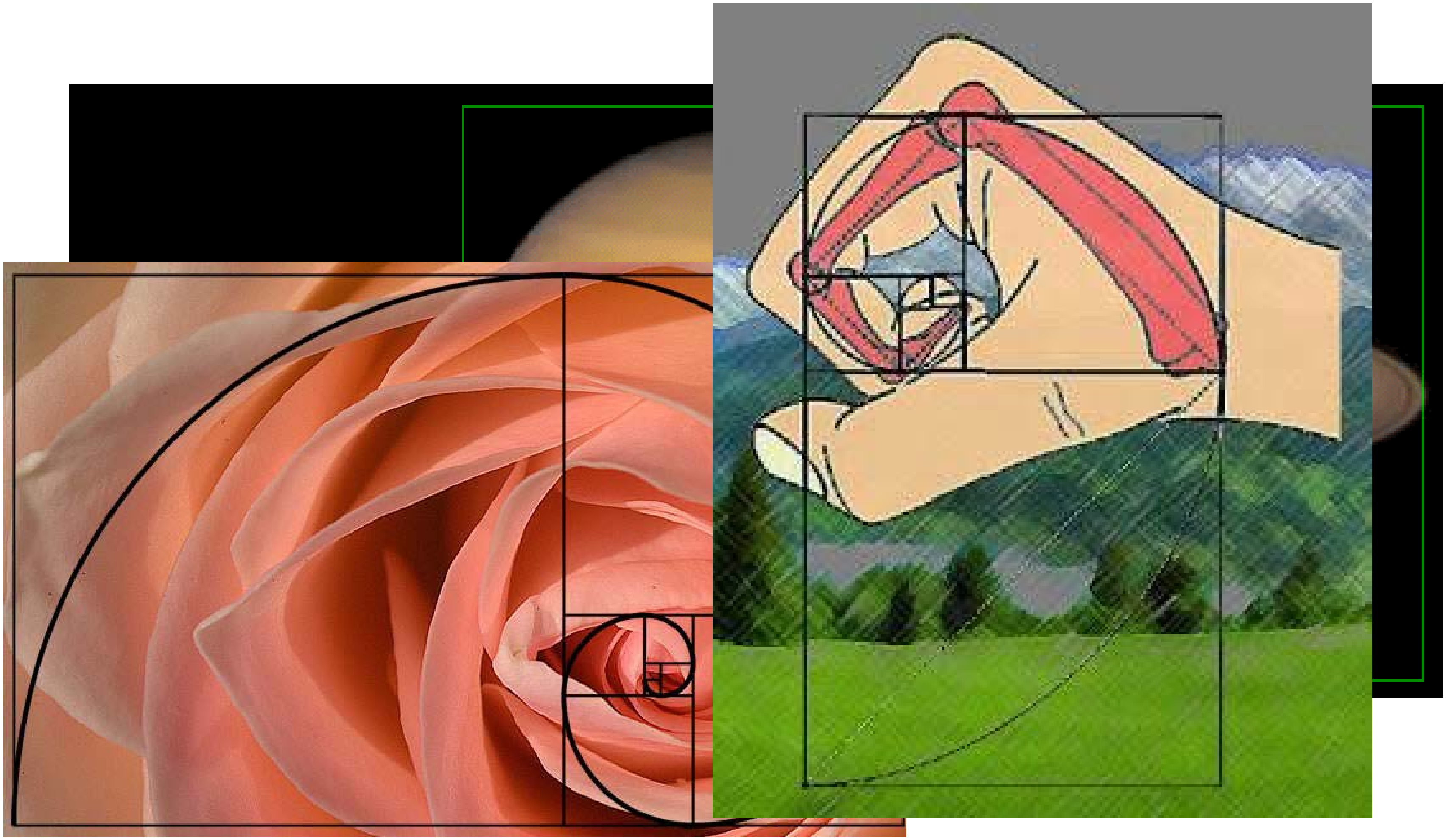
$$\frac{a+b}{a} = \frac{a}{b} = \varphi$$

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

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

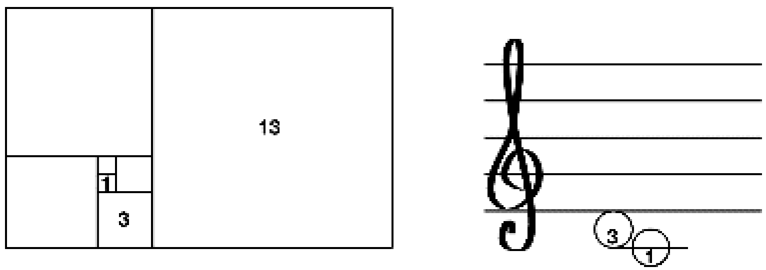
$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$$

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

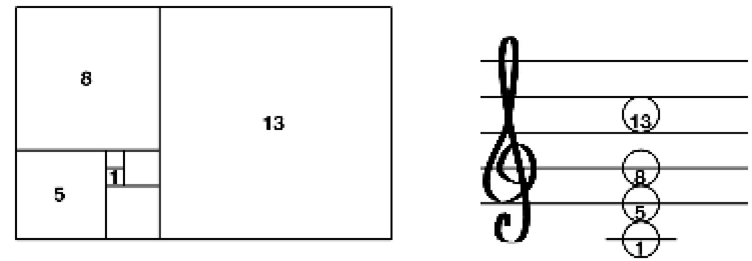


$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ \dots$$

Whole Step



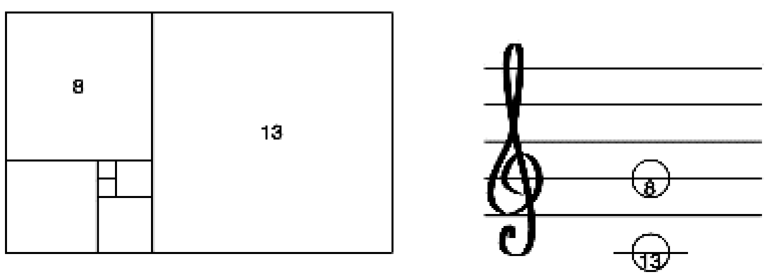
Major Triad



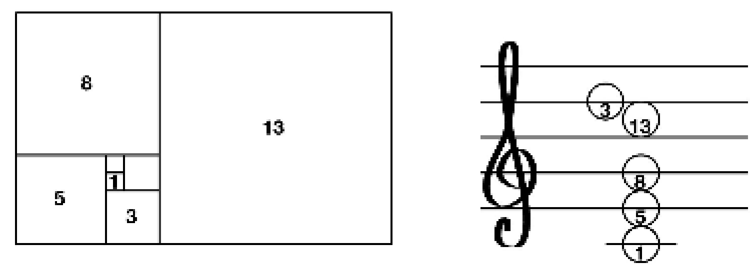
IDEAL PROPORTIONS

IDEAL CHORD STRUCTURE

Perfect Fifth



Major 9



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

The greatest of luthiers, Stradivarius, designed his violins around the golden ratio (ϕ). 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]:

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

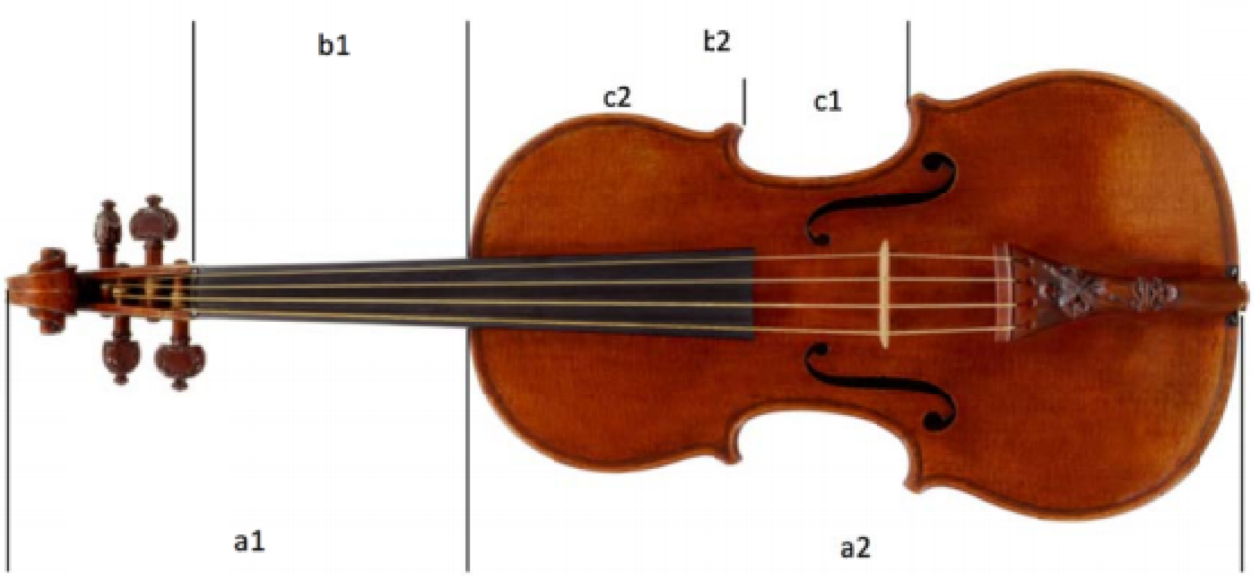


Figure 2. Photo of "Lady Blunt" Stradivarius violin (sold for nearly $16M).

φ to 3,600 digits

1.61803398874989484820458683436563811772030917980576286213544862270526046281890244970720720418939113748475408807538689175212663386222353693179318006076672635443338908659593958290563832266131992829026788067520876689250171169620703222104321626954862629631361443814975870122034080588795445474924618569536486444924104432077134494704956584678850987433944221254487706647809158846074998871240076521705751797883416625624940758906970400028121042762177111777805315317141011704666599146697987317613560067087480710131795236894275219484353056783002287856997829778347845878228911097625003026961561700250464338243776486102838312683303724292675263116533924731671112115881863851331620384005222165791286675294654906811317159934323597349498509040947621322298101726107059611645629909816290555208524790352406020172799747175342777592778625619432082750513121815628551222480939471234145170223735805772786160086883829523045926478780178899219902707769038953219681986151437803149974110692608867429622675756052317277752035361393621076738937645560606059216589466759551900400555908950229530942312482355212212415444006470340565734797663972394949946584578873039623090375033993856210242369025138680414577995698122445747178034173126453220416397232134044449487302315417676893752103068737880344170093954409627955898678723209512426893557309704509595684401755519881921802064052905518934947592600734852282101088194644544222318891319294689622002301443770269923007803085261180754519288770502109684249362713592518760777884665836150238913493333122310533923213624319263728910670503399282265263556209029798642472759772565508615487543574826471814145127000602389016207773224499435308899909501680328112194320481964387675863314798571911397815397807476150772211750826945863932045652098969855567814106968372884058746103378105444390943683583581381131168993855576975484149144534150912954070050194775486163075422641729394680367319805861833918328599130396072014455950449779212076124785645916160837059498786006970189409886400764436170933417270919143365013715766011480381430626238051432117348151005590134561011800790506381421527093085880928757034505078081454588199063361298279814117453392731208092897279222132980642946878242748740174505540677875708323731097591511776297844328474790817651809778726841611763250386121129143683437670235037111633072586988325871033632223810980901211019899176841491751233134015273384383723450093478604979294599158220125810459823092552872124137043614910205471855496118087642657651106054588147560443178479858453973128630162544876114852021706440411166076695059775783257039511087823082710647893902111569103927683845386333321565829659773103436032322545743637204124406408882673758433953679593123221343732099574988946995656473600729599983912881031974263125179714143201231127955189477817269141589117799195648125580018455065632952859859100090862180297756378925999164994642819302229355234667475932695165421402109136301819472270789012208728736170734864999815625547281137347987165695274890081443840532748378137824669174442296349147081570073525457070897726754693438226195468615331209533579238014609273510210119190218360675097308957528957746814229543394385493155339630380729169175846101460995055064803679304147236572039860073550760902317312501613204843583648177048481810991602442523271672190189334596378608787528701739359303013359011237102391712659047026349402830766876743638651327106280323174069317334482343564531850581353108549733350759966778712449058363675413289086240632456395357212524261170278028656043234942837301725574405837278267996031739364013287627701243679831144643694767053127249241047167001382478312865650649343418039004101780533950587724586655755229391582397084177298337282311152

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;
}
```

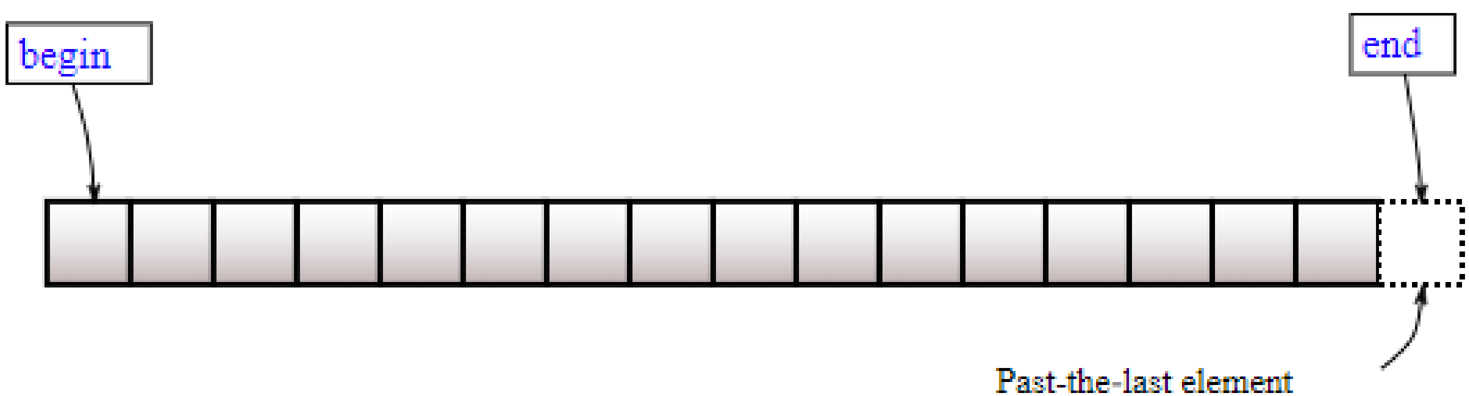
```
vector<int> EncodeCF(double x)
{
    vector<int> terms;
    while(terms.size() < 20)
    {
        terms.push_back(floor(x));
        x = x - floor(x);
        if (x < 1e-9) break;
        x = 1/x;
    }
    return NormalizeCF(terms);
}
```

x	floor(x)	x = x - 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

```cpp
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;
}
```

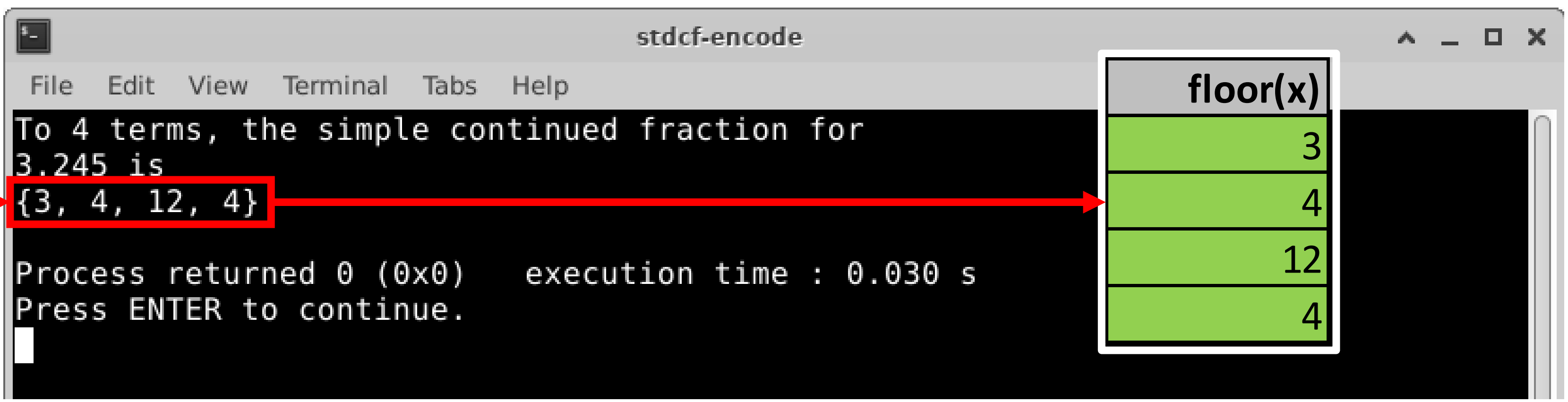
```cpp
void DisplayCF(const vector<int>& terms)
{
    cout << "{";
    auto itr = terms.begin();
    while (true)
    {
        cout << *itr;
        if (++itr == terms.end()) break;
        cout << ", ";
    }
    cout << "}\n";
    return;
}
```



Run Lab 1 – Simple CF Encoding

```cpp
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;
}
```

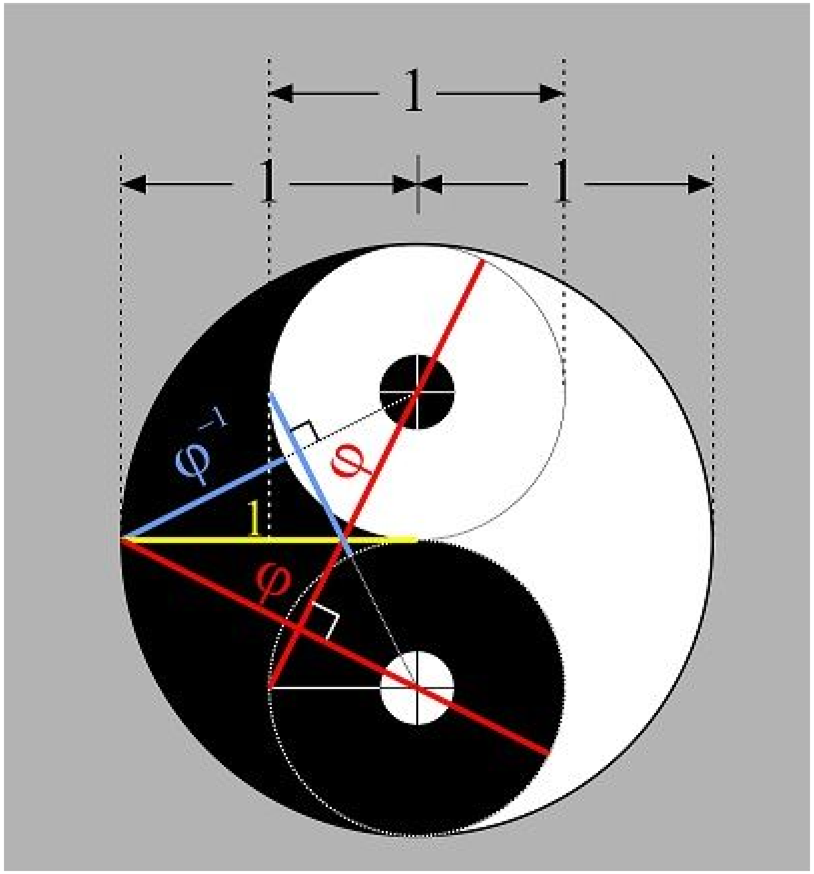
```cpp
void DisplayCF(const vector<int>& terms)
{
    cout << "{";
    auto itr = terms.begin();
    while (true)
    {
        cout << *itr;
        if (++itr == terms.end()) break;
        cout << ", ";
    }
    cout << "}\n";
    return;
}
```



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

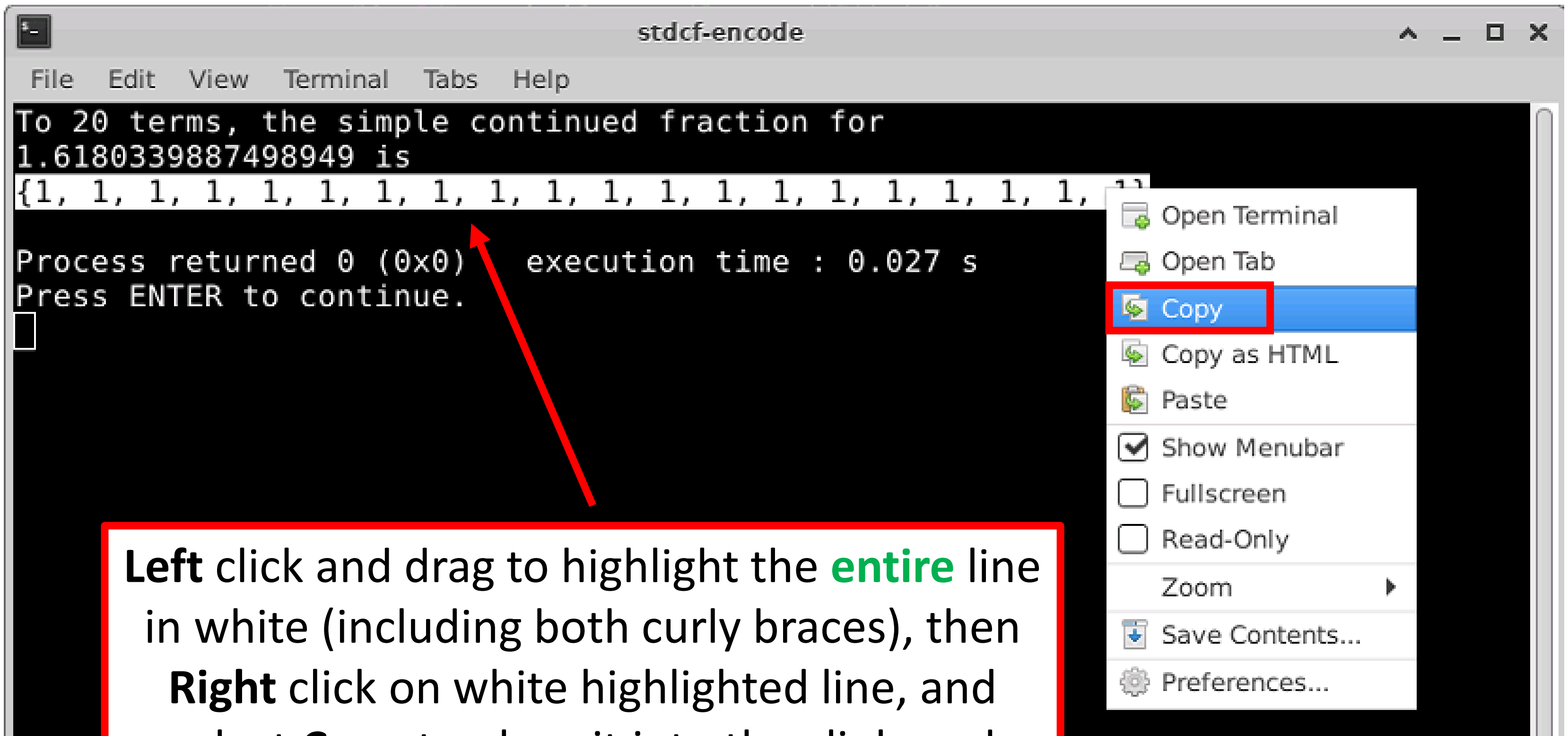
File Edit View Terminal Tabs Help

```
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

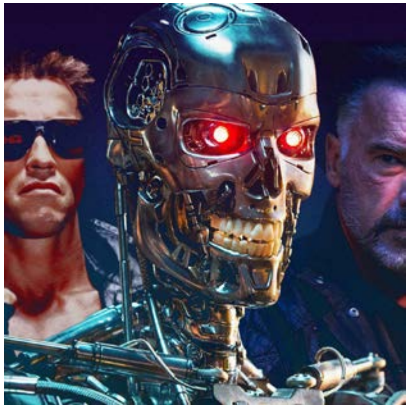


Left click and drag to highlight the **entire** line in white (including both curly braces), then **Right** click on white highlighted line, and select **Copy** to place it into the clipboard

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 <u>can</u> 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

$$h_n = a_n h_{(n-1)} + b_{(n-1)} h_{(n-2)}$$
$$k_n = a_n k_{(n-1)} + b_{(n-1)} k_{(n-2)}$$

Run Lab 2 – Simple CF Decoding

stdcf-decode

File Edit View Terminal Tabs Help

```
     1              1              1    1.00000000000000
     1              2              1    2.00000000000000
     1              3              2    1.50000000000000
     1              5              3    1.66666666666667
     1              8              5    1.60000000000000
     1             13              8    1.62500000000000
     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
                  377            233    1.61802575107296
                  610            377    1.61803713527851
                  987            610    1.61803278688525
                 1597            987    1.61803444782168
                 2584           1597    1.61803381340013
     1           4181           2584    1.61803405572755
     1           6765           4181    1.61803396316671
     1          10946           6765    1.61803399852180

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

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

π 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
847831235526582131449576857262433441893039686426243410773226978028073189154411010446823252716201052652272111
660396665573092547110557853763466820653109896526918620564769312570586356620185581007293606598764861179104533
488503461136576867532494416680396265797877185560845529654126654085306143444318586769751456614068007002378776
591344017127494704205622305389945613140711270004078547332699390814546646458807972708266830634328587856983052
358089330657574067954571637752542021149557615814002501262285941302164715509792592309907965473761255176567513
575178296664547791745011299614890304639947132962107340437518957359614589019389713111790429782856475032031986
915140287080859904801094121472213179476477726224142548545403321571853061422881375850430633217518297986622371
721591607716692547487389866549494501146540628433663937900397692656721463853067360965712091807638327166416274
888800786925602902284721040317211860820419000422966171196377921337575114959501566049631862947265473642523081
770367515906735023507283540567040386743513622224771589150495309844489333096340878076932599397805419341447377
441842631298608099888687413260472156951623965864573021631598193195167353812974167729478672422924654366800980
676928238280689964004824354037014163149658979409243237896907069779422362508221688957383798623001593776471651
228935786015881617557829735233446042815126272037343146531977774160319906655418763979293344195215413418994854
447345673831624993419131814809277771038638773431772075456545322077709212019051660962804909263601975988281613
323166636528619326686336062735676303544776280350450777235547105859548702790814356240145171806246436267945612
753181340783303362542327839449753824372058353114771199260638133467768796959703098339130771098704085913374641
442822772634659470474587847787201927715280731767907707157213444730605700733492436931138350493163128404251219
256517980694113528013147013047816437885185290928545201165839341965621349143415956258658655705526904965209858
033850722426482939728584783163057777560688876446248246857926039535277348030480290058760758251047470916439613
626760449256274204208320856611906254543372131535958450687724602901618766795240616342522577195429162991930645
537799140373404328752628889639958794757291746426357455254079091451357111369410911939325191076020825202618798
531887705842972591677813149699009019211697173727847684726860849003377024242916513005005168323364350389517029
893922334517220138128069650117844087451960121228599371623130171144484640903890644954440061986907548516026327
505298349187407866808818338510228334508504860825039302133219715518430635455007668282949304137765527939751754
613953984683393638304746119966538581538420568533862186725233402830871123282789212507712629463229563989898935
821167456270102183564622013496715188190973038119800497340723961036854066431939509790190699639552453005450580
685501956730229219139339185680344903982059551002263535361920419947455385938102343955449597783779023742161727
111723643435439478221818528624085140066604433258885698670543154706965747458550332323342107301545940516553790
686627333799585115625784322988273723198987571415957811196358330059408730681216028764962867446047746491599505
497374256269010490377819868359381465741268049256487985561453723478673303904688383436346553794986419270563872
931748723320837601123029911367938627089438799362016295154133714248928307220126901475466847653576164773794675
200490757155527819653621323926406160136358155907422020203187277605277219005561484255518792530343513984425322
341576233610642506390497500865627109535919465897514131034822769306247435363256916078154781811528436679570611
086153315044521274739245449454236828860613408414863776700961207151249140430272538607648236341433462351897576
645216413767969031495019108575984423919862916421939949072362346468441173940326591840443780513338945257423995
082965912285085558215725031071257012668302402929525220118726767562204154205161841634847565169998116141010029
960783869092916030288400269104140792886215078424516709087000699282120660418371806535567252532567532861291042
487761825829765157959847035622262934860034158722980534989650226291748788202734209222245339856264766914905562
842503912757710284027998066365825488926488025456610172967026640765590429099456815065265305371829412703369313
785178609040708667114965583434347693385781711386455873678123014587687126603489139095620099393610310291616152
881384379099042317473363948045759314931405297634757481193567091101377517210080315590248530906692037671922033
229094334676851422144773793937517034436619910403375111735471918550464490263655128162288244625759163330391072
253837421821408835086573917715096828874782656995995744906617583441375223970968340800535598491754173818839994
469748676265516582765848358845314277568790029095170283529716344562129640435231176006651012412006597558512761
785838292041974844236080071930457618932349229279650198751872127267507981255470958904556357921221033346697499
235630254947802490114195212382815309114079073860251522742995818072471625916685451333123948049470791191532673
430282441860414263639548000448002670496248201792896476697583183271314251702969234889627668440323260927524960
357996469256504936818360900323809293459588970695365349406034021665443755890045632882250545255640564482465151
875471196218443965825337543885690941130315095261793780029741207665147939425902989695946995565761218656196733
786236256125216320862869222103274889218654364802296780705765615144632046927906821207388377814233562823608963
208068222468012248261177185896381409183903673672220888321513755600372798394004152970028783076670944474560134
556417254370906979396122571429894671543578468788614445812314593571984922528471605049221242470141214780573455
105008019086996033027634787081081754501193071412233908663938339529425786905076431006383519834389341596131854
347546495569781038293097164651438407007073604112373599843452251610507027056235266012764848308407611830130527
932054274628654036036745328651057065874882256981579367897669742205750596834408697350201410206723585020072452
256326513410559240190274216248439140359989535394590944070469120914093870012645600162374288021092764579310657
922955249887275846101264836999892256959688159205600101655256375678

Continued Fractions

What is the simple CF encoding for π ?

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

π = [3;7,15,1,292,1,1,1,2,1,3,1,14,3,3,23,1,1,7….]

(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

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

In a <u>generalized</u> continued fraction,
a_n and b_n can now be *any* expression

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

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

Generalized Continued Fractions

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

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

Euler

$$\pi = [\overset{a_0}{3};\ \overset{b_0}{1},\ \{\overset{b_n}{(2n+1)^2} \,|\, \overset{a_n}{6}\}]$$

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

Generalized Continued Fractions

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

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	[illegible]
6	6	169	2971101	945945	3.140881341	[illegible]
7	6	225	50952465	16216200	3.142071817	-0.0004791[illegible]
8	6	289	9.74E+08	3.1E+08	3.141254824	0.0003378[illegible]
9	6	361	2.06E+10	6.55E+09	3.141839619	-0.000246965

Gen CFs converge slowly

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

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

Generalized Continued Fractions

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

Biersach

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

$$\underset{\substack{\uparrow\\ a_0}}{} \quad \underset{\substack{\uparrow\\ b_0}}{} \quad \underset{\substack{\uparrow\\ b_n}}{} \quad \underset{\substack{\uparrow\\ a_n}}{}$$

π = [3; 1, {(8n²-7)|(8n-1)}]

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

Generalized Continued Fractions

What is a **generalized** CF expansion for π ?

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 $\boldsymbol{\pi}$?

Biersach

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

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

$\boldsymbol{\pi}$ **= [2; 8, {(4n²+8n)|(4n+2)}]**

Generalized Continued Fractions

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

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

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

My GCF for $\boldsymbol{\pi}$ converges faster than Euler's! ☺

Generalized Continued Fractions

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

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

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

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

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 $\boldsymbol{x}$ & $\boldsymbol{y}$ for a given $\boldsymbol{n}$ (assume $x, y, \boldsymbol{n} \in \mathbb{Z}^{+}$) such that:

$$x^2 - \boldsymbol{n}y^2 = 1$$

- 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 $\boldsymbol{x}$ & $\boldsymbol{y}$ values that solve the equation for each $\boldsymbol{n}$ 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

Run Lab 3
Pell's Equation

Hyphens indicate no solution was found in the allowed search space

File Edit View Terminal Tabs

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?

$$\boldsymbol{n} = 1, 4, 9, 16, 25, 36, 49, \mathbf{61}, 64, \ldots$$

- 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 <u>not</u> 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

Simple CF

Period = 2

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

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

Period = 5

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

Period = 5

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

Period = 11

$\sqrt{61} =$ {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

Simple CF

$\sqrt{13}$ Period = 4

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

$\sqrt{61}$ Period = 11

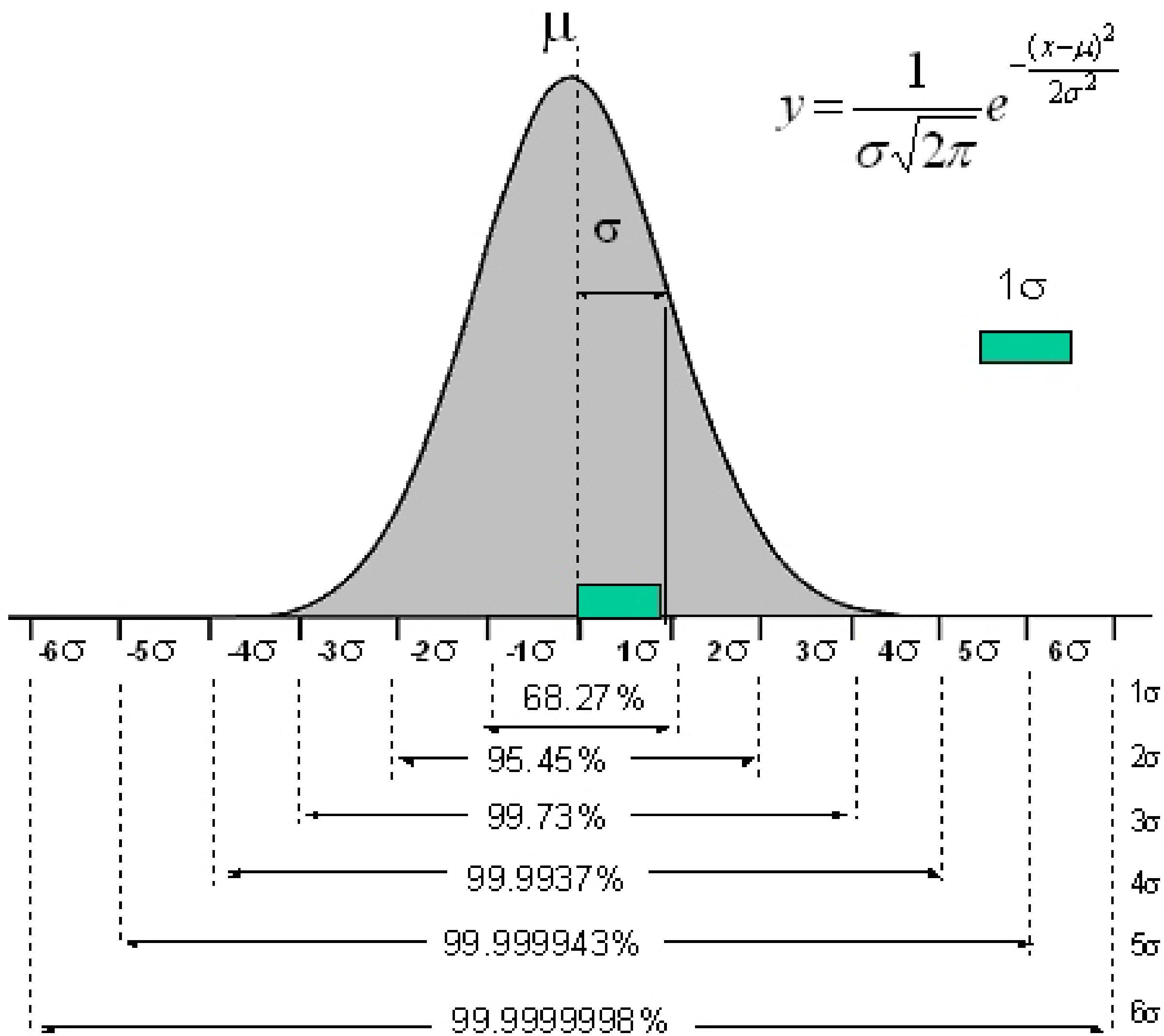
…then the x & y will be large for the solution to Pell's equation

Normal Distribution

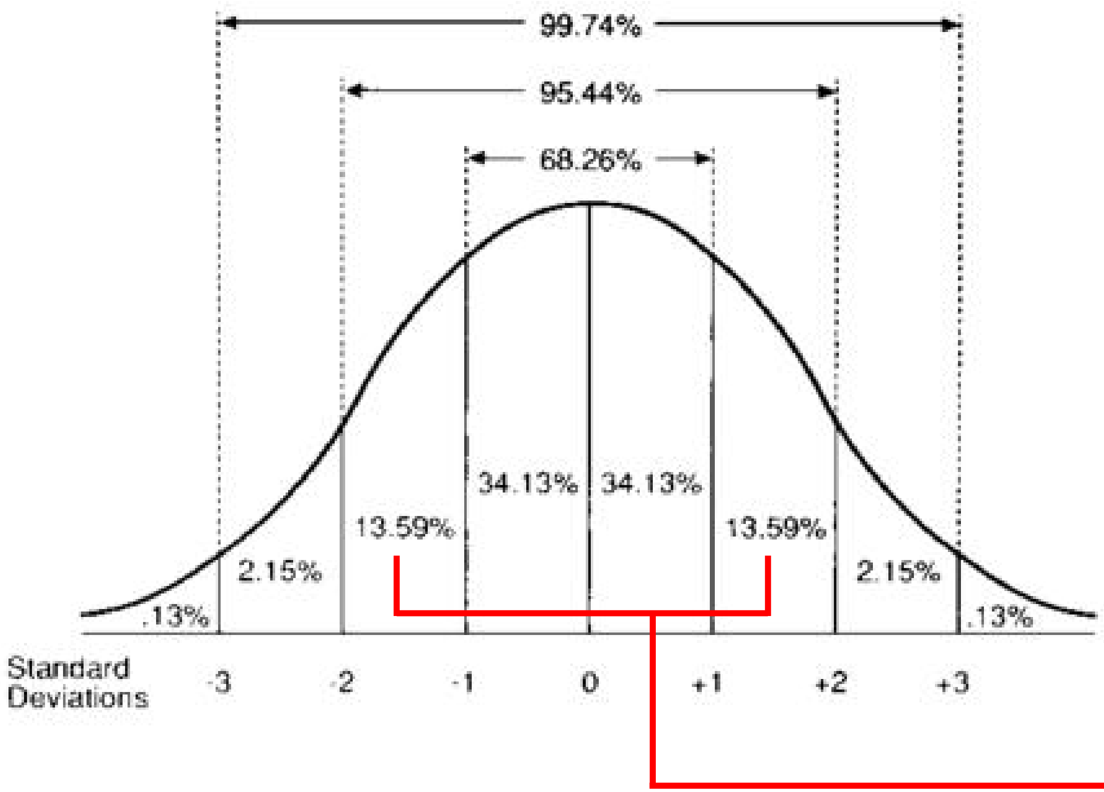
- Until recently, most computer languages only provided a **uniform** pseudo-random number generator
- Growing up I had heard of a **bell curve** and I understood the rationale for **curving** test scores
- However I could not create a **normal distribution** using my 1978 vintage TRS-80 computer using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

… or could I?

Normal Distribution



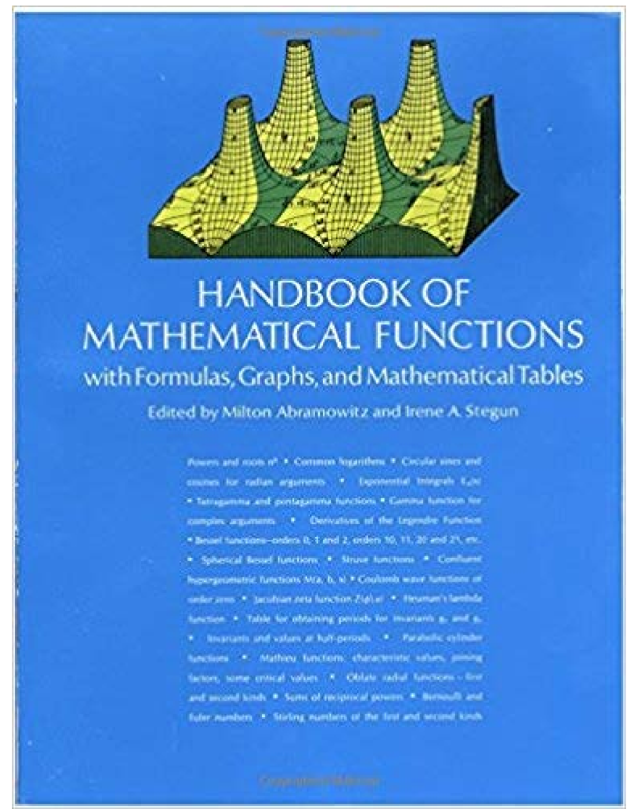
Normal Distribution



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

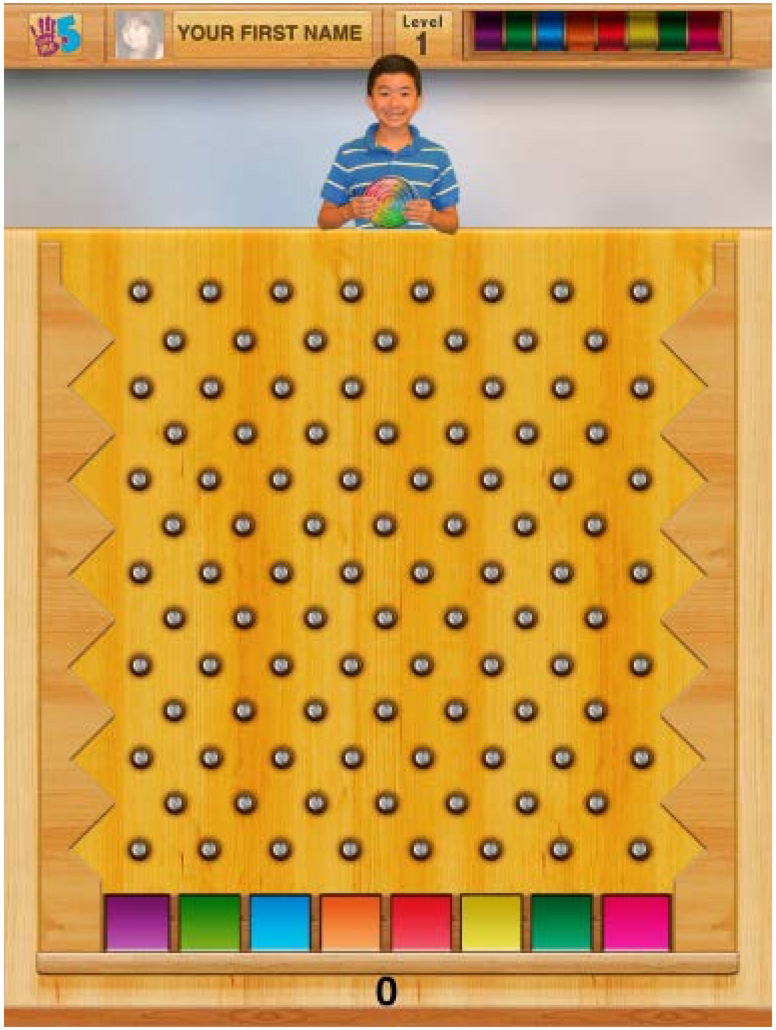


Normal Distribution

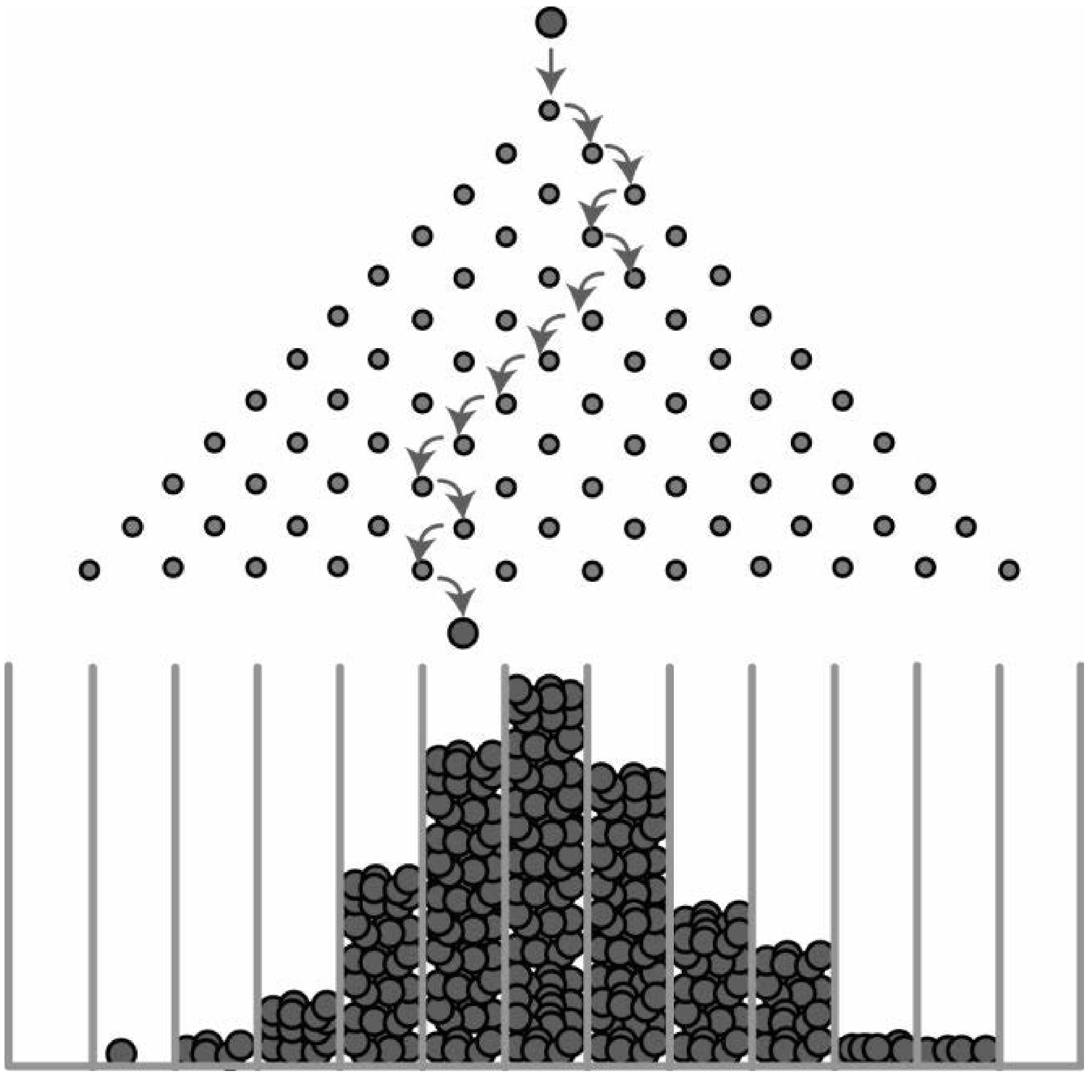
```
double StegunNormal(double mean, double stddev)
{
    double q = 1 - distUniform(generator);
    double p = (q < 0.5) ? q : 1 - q;
    double t = sqrt(log((1 / (p * p))));
    double x = t - (2.515517 + 0.802853 * t + 0.010328 * (t * t)) /
        (1 + 1.432788 * t + 0.189269 * (t * t) + 0.001308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

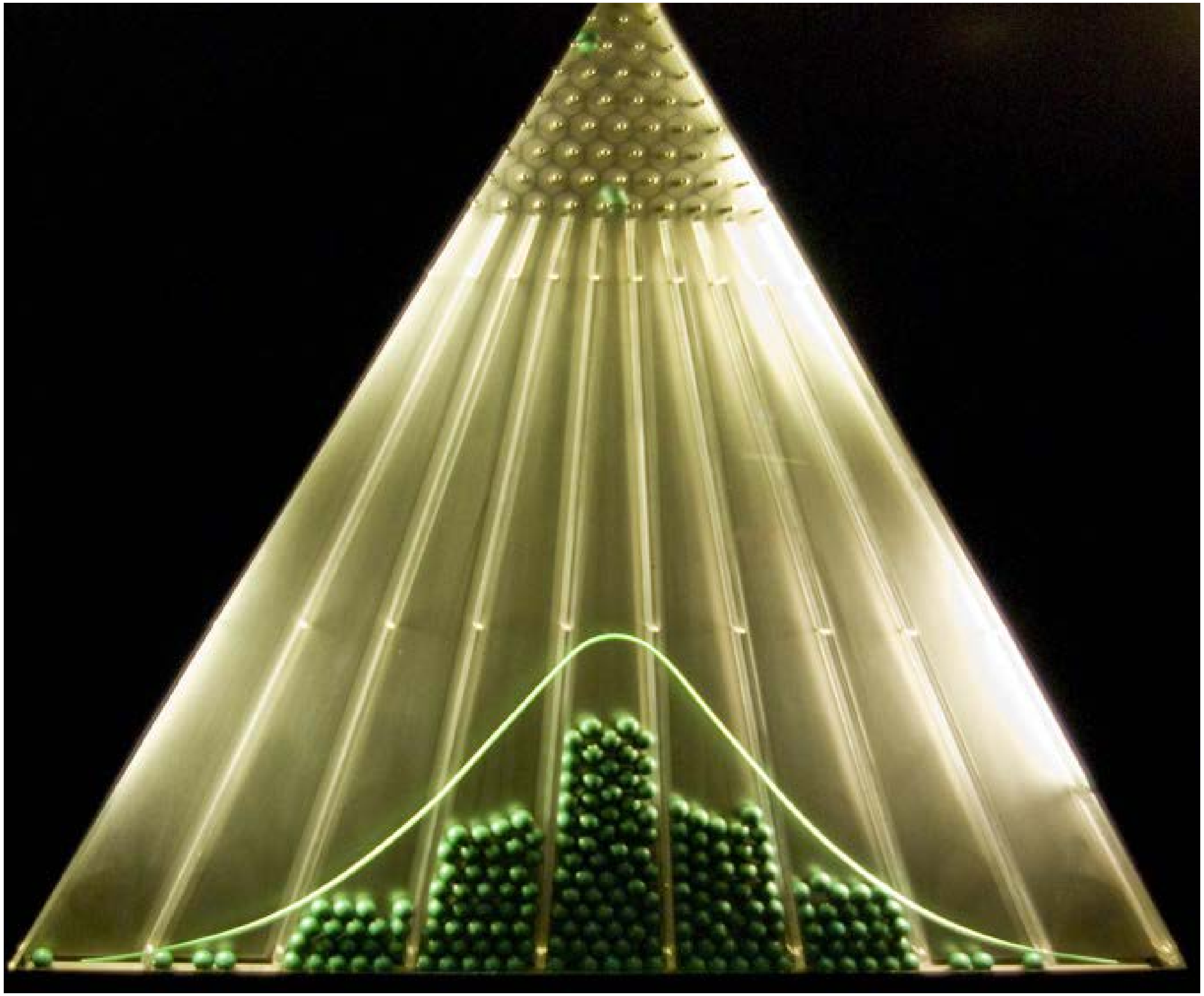
Pachinko Distribution



Pachinko Distribution



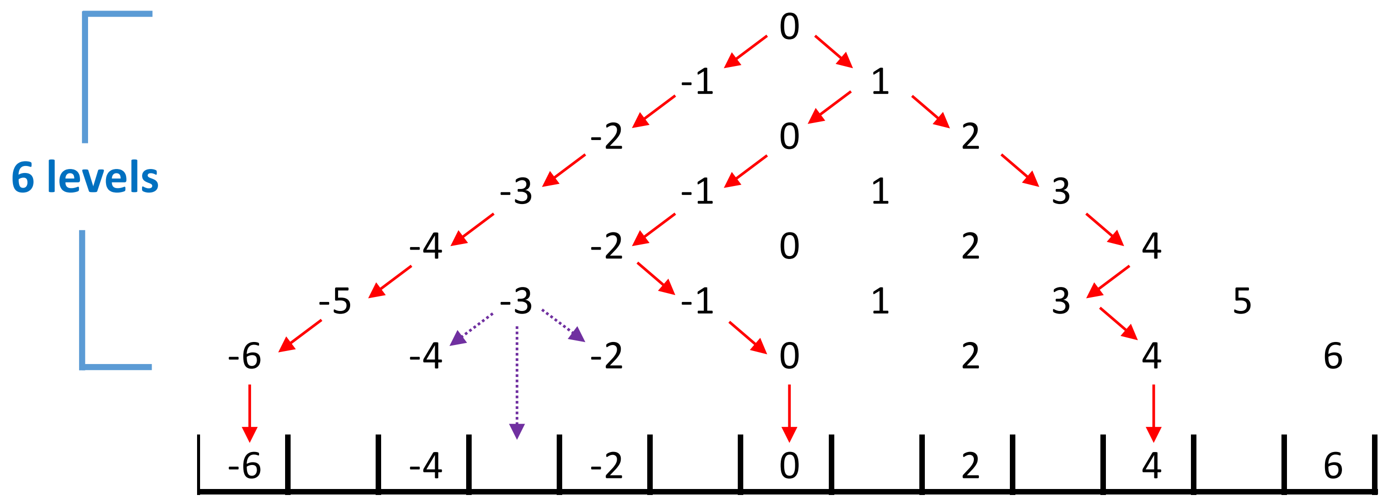
Pachinko Distribution



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




```
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);
```

The range is ½ the number of levels

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);
}
```

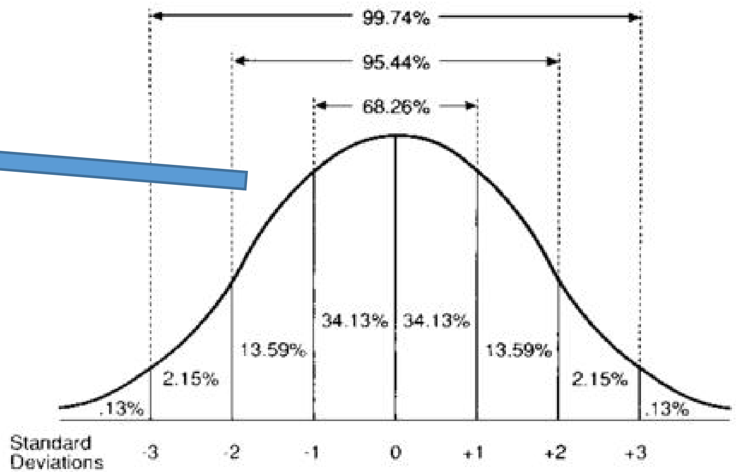
The mean slot *should be* = 0

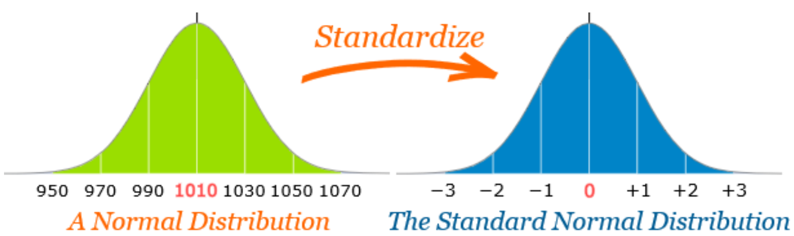
View Lab 4 – Pachinko Normal

```
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 a integral multiple of the **standard deviation**
- Each **slot** is 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
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)    exec
Press ENTER to continue.
```

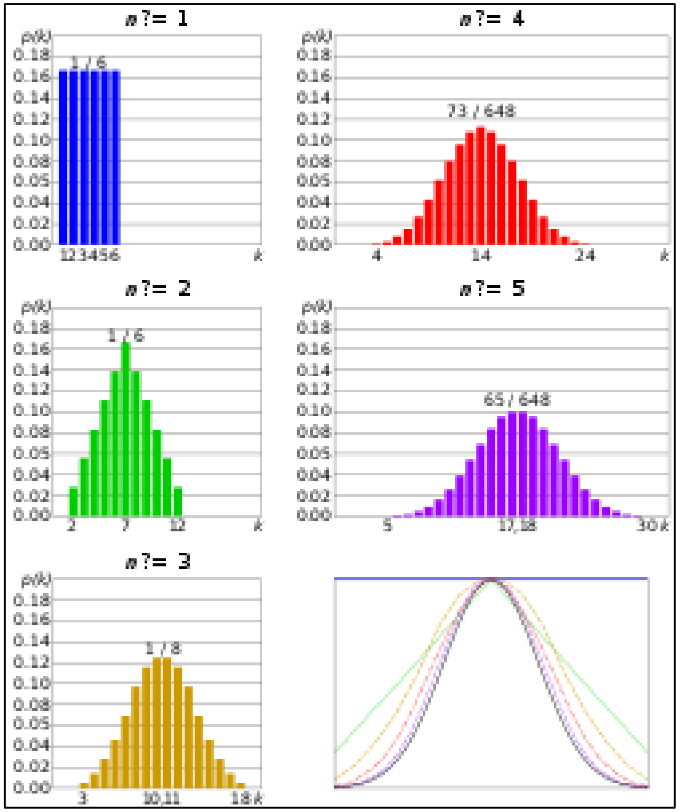
```
pachinko-normal
File  Edit  View  Terminal  Tabs  Help
 Balls: 1,000
Levels: 100

   Sigma  Pachinko     Normal
       1       644        682
       2       301        271
       3        52         43
       4         3          2

Chi-Squared: 7.822

Process returned 0 (0x0)    exec
Press ENTER to continue.
```

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

   Sigma  Pachinko     Normal
       1       662        682
       2       295        271
       3        41         43
       4         2          2

Chi-Squared: 2.805

Process returned 0 (0x0)    exec
Press ENTER to continue.
```

Central Limit Theorem

Central limit theorem

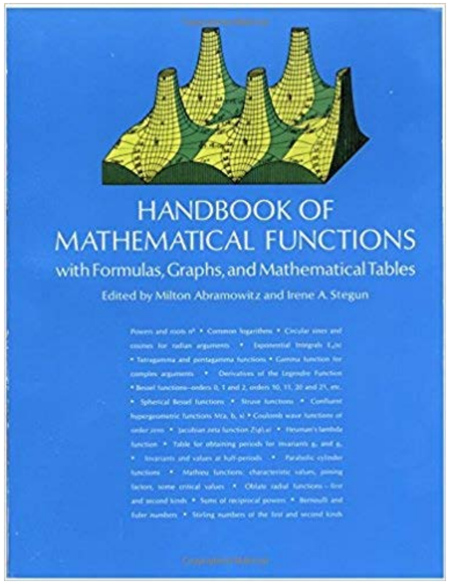
In probability theory, the **central limit theorem** (**CLT**) establishes that, for the most commonly studied scenarios, when independent random variables are added, their sum tends toward a normal distribution (commonly known as a *bell curve*) even if the original variables themselves are not normally 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.[1][2] 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';
    }
}
```

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** $\{x^2 - \boldsymbol{n}y^2 = 1\}$ is related to the period of the *simple* continued fraction of $\sqrt{\boldsymbol{n}}$
- Memorizing thousands of digits of $\boldsymbol{\pi}$ 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