Preamble

In[1]:=

In[2]:=
```
$PreRead = (# /. s_String /; StringMatchQ[s, NumberString] &&
        Precision@ToExpression@s == MachinePrecision :> s <> "`100." &);
3 / 1.5 + Pi / 7
Precision[%]

$MinPrecision = 4 * $MachinePrecision
```

Out[3]=
```
2.448798950512827605494663340468500412028167057053586545853563513186830915183744142661147832191731010
```

Out[4]=
```
100.088
```

Out[5]=
```
63.8184
```

Above commands enable us to globally work at a higher than default precision

In[6]:=
```
ϵin = 0.01;
steps = 50;
minϵpow = 25;
```

Above are global parameters. Very important is ϵin, which in the paper is also called ϵ_{raw}, where we consider the cases
ϵin=0.01 (the high noise regime)
ϵin=0.001 (the low noise regime)

Distillation Cost

Protocols

Success probabiltiy and error out for various protocols. Also a map funciton that takes as input a sqeuence, e.g. $\{\epsilon, C\}$ where ϵ is the error of inpout states and C is there cost. The function then outputs a new pair.

standard MEK_3

In[9]:=
```
PsMEK[ϵ_] = 1 - 10 ϵ + 58 ϵ^2 - 192 ϵ^3 + 400 ϵ^4 - 544 ϵ^5 + 480 ϵ^6 - 256 ϵ^7 + 64 ϵ^8;
ϵoutMEK[ϵ_] = (9 ϵ^2 - 56 ϵ^3 + 160 ϵ^4 - 256 ϵ^5 + 240 ϵ^6 - 128 ϵ^7 + 32 ϵ^8) / PsMEK[ϵ];
MapMEK[input_] := {ϵoutMEK[input[[1]]], 5 * input[[2]] / PsMEK[input[[1]]]};
```

standard Bravyi-Haah

In[12]:=
```
Pbh[ϵ_, k_] = 1/8 * (1 + x^8 + 6 x^(4+2 k)) /. x → (1 - 2 ϵ);
ϵoutbh[ϵ_, k_] = (1 - (1 + 2 x^7 + x^8 + 6 x^(3+2 k) + 6 x^(4+2 k))/(2 * (1 + x^8 + 6 x^(4+2 k)))) /. x → (1 - 2 ϵ);
MapBH[input_, k_] := {ϵoutbh[input[[1]], k], (3 + 8/k) * input[[2]]/Pbh[input[[1]], k]}
```

15 qubit Reed-Muller code

In[15]:=
```
Ps15[ϵ_] := (1 + 15 (1 - 2 ϵ)^8)/16;
Er15[ϵ_] := (1 - 15 (1 - 2 ϵ)^7 + 15 (1 - 2 ϵ)^8 - (1 - 2 ϵ)^15) / (32 * Ps15[ϵ]);
Map15[input_] := {Er15[input[[1]]], 15 * input[[2]]/Ps15[input[[1]]]};
```

MEK$_L$

In[18]:=
```
PmekL[ϵ3_, ϵk_, δ_] = 1 - 448 ϵ3^5 + 448 ϵ3^6 - 256 ϵ3^7 +
    64 ϵ3^8 - 1/2 δ (1 - 2 ϵ3)^4 (1 - 2 ϵk)^2 - 2 ϵk + 2 ϵk^2 - 64 ϵ3^3 (2 - ϵk + ϵk^2) +
    32 ϵ3^4 (9 - ϵk + ϵk^2) - 8 ϵ3 (1 - 2 ϵk + 2 ϵk^2) + 8 ϵ3^2 (5 - 6 ϵk + 6 ϵk^2);
ϵoutmekL[ϵ3_, ϵk_, δ_] = 1/PmekL[ϵ3, ϵk, δ] *
    (δ/4 - 2 δ ϵ3 + 8 ϵ3^2 + 6 δ ϵ3^2 - 48 ϵ3^3 - 8 δ ϵ3^3 + 136 ϵ3^4 + 4 δ ϵ3^4 - 224 ϵ3^5 +
      224 ϵ3^6 - 128 ϵ3^7 + 32 ϵ3^8 + ϵk^2 - δ ϵk^2 - 8 ϵ3 ϵk^2 + 8 δ ϵ3 ϵk^2 + 24 ϵ3^2 ϵk^2 -
      24 δ ϵ3^2 ϵk^2 - 32 ϵ3^3 ϵk^2 + 32 δ ϵ3^3 ϵk^2 + 16 ϵ3^4 ϵk^2 - 16 δ ϵ3^4 ϵk^2);
MapMEKLbeyond[input3_, inputk_, inputδ_] :=
  {ϵoutmekL[input3[[1]], inputk[[1]], inputδ[[1]]],
   (2 * inputk[[2]] + 8 * input3[[2]] + inputδ[[2]])/(2 * PmekL[input3[[1]], inputk[[1]], inputδ[[1]]])}
```

DP

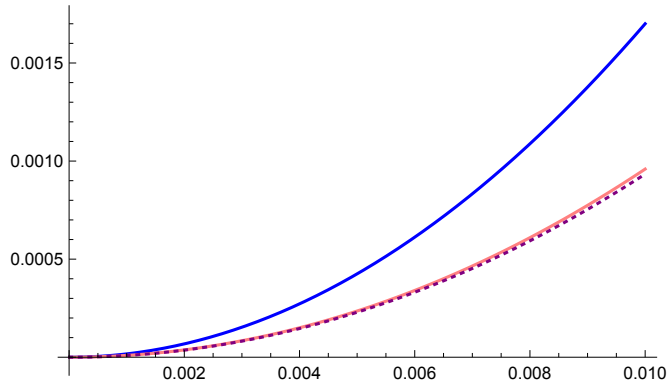
In[21]:=
```
PsDP[ϵ3_, ϵk_, δ_] = 1 - 2 * ϵk - 16 * ϵ3 - δ;
ϵoutDP[ϵ3_, ϵk_, δ_] = ϵk^2 + 15 * ϵ3^2 + δ;
```

Approximate, leading order expressions for the Duclos-Ciani and Poulin protcol. Provided by Duclos-Ciani in private communication. Note they differ from those given in their paper as the published paper gives an expression that counts all the weight 2 errors as contirbuting to ϵout, when in fact only a fraction actually contribute to ϵout. While the equation in the Duclos-Ciani and Poulin paper is incorrect, their analysis and plots are based on the correct expression obtained by explicit simulation in mathematica.

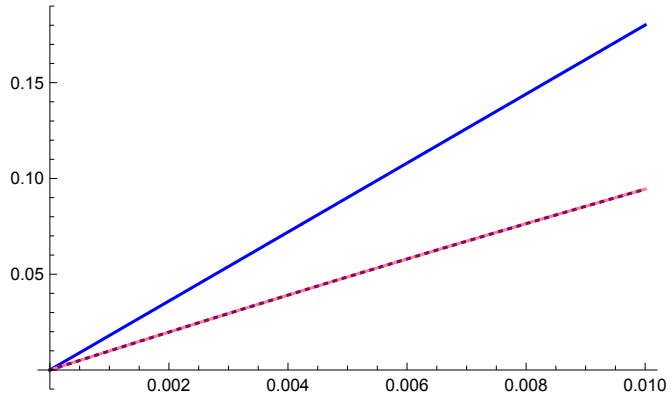
MEK performance for 1 round

In[23]:=

```
ϵplot = Plot[{ϵoutDP[ϵ, ϵ, ϵ^2], ϵoutmekL[ϵ, ϵ, ϵ^2], ϵoutMEK[ϵ]}, {ϵ, 10^-5, 0.01},
    PlotStyle → {{Solid, Blue}, {{Pink, Solid}}, {Purple, Dotted}}];
probplot = Plot[{1 - PsDP[ϵ, ϵ, ϵ^2], 1 - PmekL[ϵ, ϵ, ϵ^2], 1 - PsMEK[ϵ]}, {ϵ, 0, 0.01},
    PlotStyle → {{Solid, Blue}, {{Pink, Solid}}, {Purple, Dotted}}];
GraphicsGrid[{{ϵplot}, {probplot}}]
```



Out[25]=



Comparison of the failure probabiltiy and error out for MEL_3, MEL_L and DP_L. Equations used are exact for MEL_3 and MEL_L but only leading order approximations for DP_L.

Plots above used as basis for Fig 4.a of paper.

Functions for manipulating lists of data

In[26]:=
```
BestOf[listin_] :=
 Module[{size, best, j, ϵupper, now}, size = Dimensions[listin][[1]];
  best = {{ϵin, 1}};
  now = listin;
  For[j = 0, j < steps, j++;
   ϵupper = ϵin * 10^-(minepow*j/steps);
   now = Select[now, #[[1]] ≤ ϵupper &];
   now = SortBy[now, #[[2]] &];
   If[Dimensions[now][[1]] > 0,
    best = Append[best, now[[1]]];];
  ];
  best = DeleteDuplicates[best];
  best
 ]
```

BestOf takes large lists of the form {epsilon, Cost} and sifts for the lowest cost protocols within a certain epsilon window.

In[27]:=
```
nextPIV[oldPIV_, MagicCost_] :=
 Module[{tempCost, dimOLD, dimNEW, j, k, jkelement, δ, C, ϵnew, ϵold},
  tempCost = {{ϵin, 1}};
  dimOLD = Dimensions[oldPIV][[1]];
  dimNEW = Dimensions[MagicCost][[1]];
  tempCost = {{ϵin, 1}};
  For[j = 0, j < dimOLD, j++;
   For[k = 0, k < dimNEW, k++;
    ϵnew = MagicCost[[k, 1]];
    ϵold = oldPIV[[j, 1]];
    δ = 1/2 * ϵnew + 1/2 * (ϵnew * (1 - ϵold) + ϵold * (1 - ϵnew));
    C = MagicCost[[k, 2]] + 1/2 * oldPIV[[j, 2]];
    jkelement = {δ, C};
    tempCost = Append[tempCost, jkelement];
   ];];
  tempCost = BestOf[tempCost];
  Print["Pivotal compile complete with output of size =>",
   Dimensions[tempCost][[1]]];
  tempCost]
```

nestPIV outputs a cost function C(δ_L, P_L) for a pivotal rotation based on the oldPIV C(δ_{L-1}, P_{L-1}) and the relevant Magic Cost C(ϵ_L, W_L).

In[28]:=
```
nextRound[cost3_, costk_ , costPIV_] :=
 Module[{size3, sizek, sizePIV, n3, nk, nPIV, tempCost}, tempCost = costk;
  size3 = Dimensions[cost3][[1]];
  sizek = Dimensions[costk][[1]];
  sizePIV = Dimensions[costPIV][[1]];
  For[n3 = 0, n3 < size3, n3++;
   For[nk = 0, nk < sizek, nk++;
    For[nPIV = 0, nPIV < sizePIV, nPIV++;
     tempCost = Append[tempCost,
       MapMEKLbeyond[cost3[[n3]], costk[[nk]], costPIV[[nPIV]]]];
    ];
   ];
  ];
  tempCost = BestOf[tempCost];
  Print["round complete with output of size =>", Dimensions[tempCost][[1]]];
  tempCost
 ]
```

nextRound takes as input a list of many achieveable values of {ϵ, C} for various magic states and then applies another round of distillation of MEK_L to obtain another such list. nextRound is entirely brute force and searches through all possible combinations of different inputs. This is a huge number of combinations so the function ends with BestOf to select just the most efficient elements of the list.

In[29]:=
```
Filter[list_] := Select[list, 10^-20 < #[[1]] < ϵin/2 &];
MakePlotList[TAB_] := Module[{temp, sizeof, j, temp2},
  temp2 = Filter[TAB];
  sizeof = Dimensions[temp2][[1]];
  temp = {{ -Log[10, temp2[[1, 1]]]  , temp2[[1, 2]]}};
  For[j = 1, j ≤ sizeof, j++,
   temp = Append[temp, { -Log[10, temp2[[j, 1]]]  , temp2[[j, 2]]} ];
  ];
  temp
 ]
```

The above functions are simply used to filter the data and convert the Cost into a Log(cost) and is used before plotting.

Many rounds

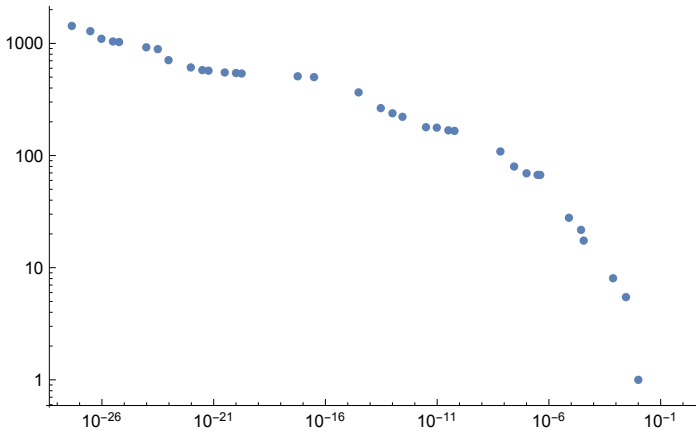
Construct COST menu for level=3 states

In[31]:=
```
nextRoundlev3[CostIn_] := Module[{Halfk, j, Level3Cost, sizeof},
  Level3Cost = CostIn;
  sizeof = Dimensions[Level3Cost][[1]];
  For[j = 0, j < sizeof, j++;
   Level3Cost = Append[Level3Cost, Map15[Level3Cost[[j]]]];
   Level3Cost = Append[Level3Cost, MapMEK[Level3Cost[[j]]]];
   For[Halfk = 0, Halfk < 100, Halfk++;
    Level3Cost = Append[Level3Cost, MapBH[ Level3Cost[[j]], 2 * Halfk ]];];
   Level3Cost = Delete[Level3Cost, j];
  ];
  BestOf[Level3Cost]
 ]
```

In[32]:=
```
Level3Cost = {{ϵin, 1}};
Level3Cost = nextRoundlev3[Level3Cost];
Level3Cost = nextRoundlev3[Level3Cost];
Level3Cost = nextRoundlev3[Level3Cost];
Level3Cost = nextRoundlev3[Level3Cost];
Level3Cost = nextRoundlev3[Level3Cost];
L3 = ListLogLogPlot[Level3Cost, Ticks → Automatic]
Min[Level3Cost]
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[38]=



Out[39]= $4.488907775454651604244394582624565596415934499169621245923646276342827 53 \times 10^{-28}$

Construct COST menu for level=4 states

In[40]:=
```
MapMEKLbeyond[{ϵin, 1}, {ϵin, 1}, {ϵin, 1}]
```

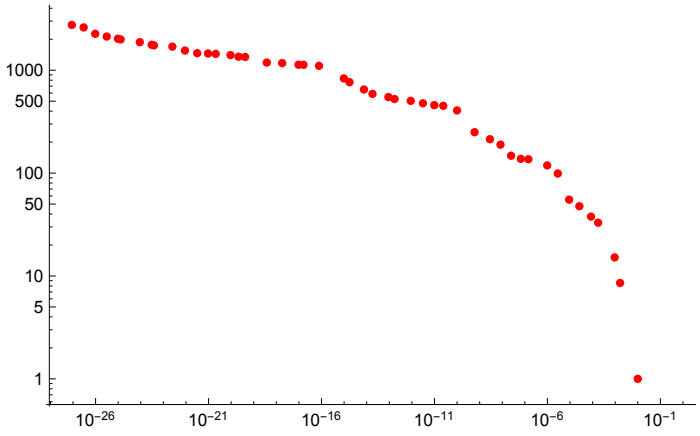
Out[40]=
```
{0.00349604198026938989013862919628895556082813251501314759871201416467256057↘
  85093873471116196533665671,
 6.10309074002231590835893725314436315591597622255779740016964167023119036848↘
  78989167148674033353763 52}
```

In[41]:=
```
Level4Cost = {MapMEKLbeyond[{ϵin, 1}, {ϵin, 1}, {ϵin, 1}]};
Level4Cost = nextRound[Level3Cost, Level4Cost , Level3Cost];
Level4Cost = nextRound[Level3Cost, Level4Cost , Level3Cost];
Level4Cost = nextRound[Level3Cost, Level4Cost , Level3Cost];
Level4Cost = nextRound[Level3Cost, Level4Cost , Level3Cost];
L4 = ListLogLogPlot[Level4Cost, PlotStyle → Red]
```

```
round complete with output of size =>6
round complete with output of size =>15
round complete with output of size =>31
round complete with output of size =>48
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[46]=



In[47]:=

In[48]:=

Construct COST for level=5 states

In[49]:=
```
Level5PIV = nextPIV[Level3Cost, Level4Cost];
```

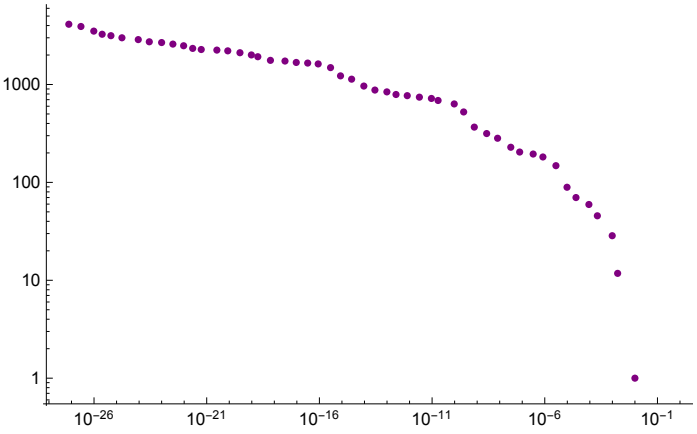
```
Pivotal compile complete with output of size =>48
```

In[50]:=
```
Level5Cost = {MapMEKLbeyond[{ϵin, 1}, {ϵin, 1}, {ϵin, 1}]};
Level5Cost = nextRound[Level3Cost, Level5Cost , Level5PIV];
Level5Cost = nextRound[Level3Cost, Level5Cost , Level5PIV];
Level5Cost = nextRound[Level3Cost, Level5Cost , Level5PIV];
Level5Cost = nextRound[Level3Cost, Level5Cost , Level5PIV];
L5 = ListLogLogPlot[Level5Cost, PlotStyle → Purple]
```

```
round complete with output of size =>6

round complete with output of size =>15

round complete with output of size =>33

round complete with output of size =>51
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[55]=



Construct COST for level=6 states

In[56]:=
```
Level6PIV = nextPIV[ Level5PIV, Level5Cost];
```

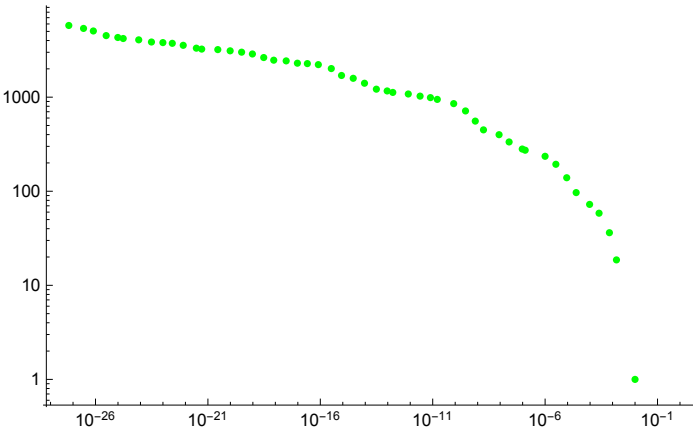
```
Pivotal compile complete with output of size =>50
```

In[57]:=
```
Level6Cost = {MapMEKLbeyond[{ϵin, 1}, {ϵin, 1}, {ϵin, 1}]};
Level6Cost = nextRound[Level3Cost, Level6Cost , Level6PIV];
Level6Cost = nextRound[Level3Cost, Level6Cost , Level6PIV];
Level6Cost = nextRound[Level3Cost, Level6Cost , Level6PIV];
Level6Cost = nextRound[Level3Cost, Level6Cost , Level6PIV];
L6 = ListLogLogPlot[Level6Cost, PlotStyle → Green]
```

```
round complete with output of size =>6

round complete with output of size =>15

round complete with output of size =>33

round complete with output of size =>51
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[62]=



Construct COST for level=7 states

In[63]:=
```
Level7PIV = nextPIV[Level6PIV, Level6Cost ];
```

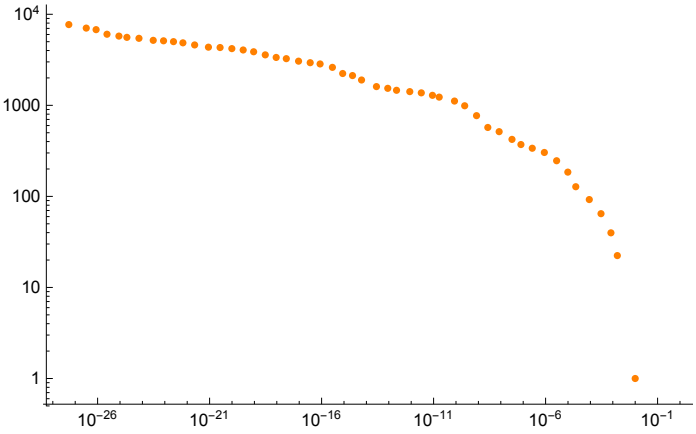
```
Pivotal compile complete with output of size =>50
```

In[64]:=
```
Level7Cost = {MapMEKLbeyond[{ϵin, 1}, {ϵin, 1}, {ϵin, 1}]};
Level7Cost = nextRound[Level3Cost, Level7Cost , Level7PIV];
Level7Cost = nextRound[Level3Cost, Level7Cost , Level7PIV];
Level7Cost = nextRound[Level3Cost, Level7Cost , Level7PIV];
Level7Cost = nextRound[Level3Cost, Level7Cost , Level7PIV];
L7 = ListLogLogPlot[Level7Cost, PlotStyle → Orange]
```

```
round complete with output of size =>6

round complete with output of size =>15

round complete with output of size =>33

round complete with output of size =>51
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[69]=

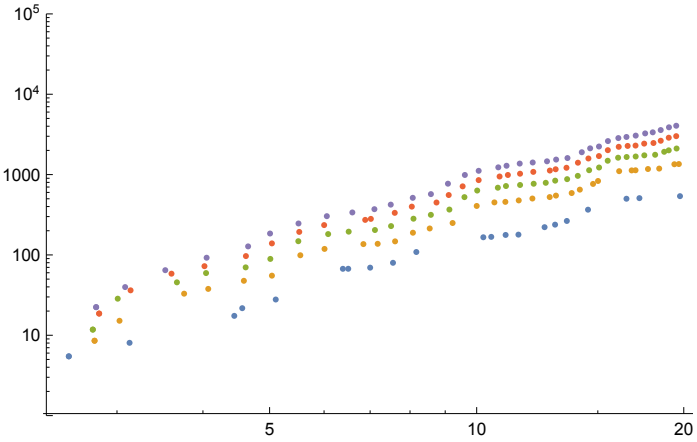


Build combined plots from data

In[70]:=
```
mek3 = MakePlotList[Level3Cost];
mek4 = MakePlotList[Level4Cost];
mek5 = MakePlotList[Level5Cost];
mek6 = MakePlotList[Level6Cost];
mek7 = MakePlotList[Level7Cost];

mekALL = ListLogLogPlot[{mek3, mek4, mek5, mek6, mek7}, PlotRange → {1, 10^5}]
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[75]=



Construct a plot on a Log vs LogLog scale showing all W_L distillation costs

Compare with DP protocol

In[126]:=
```
CP4 = {{3.46, 49.3}, {6.83, 233}, {13.6, 1110}};
CP5 = {{3.50, 73.2}, {6.78, 351}, {13.5, 1700}};
CP6 = {{3.53, 97.4}, {6.76, 482}, {13.4, 2360}};
CP7 = {{3.55, 122}, {6.75, 626} , {13.3, 3100}};
```

Above data taken from published DP paper in table 1.

In[104]:=
```
Plotter[CPin_, mekin_, ymax_] :=
 Module[{α, β, γ, p1, p2, p3, p4, CPfitted, mekfitted, CPsol, meksol},
  CPsol = FindFit[CPin, α + β * x^γ, {α, β, γ}, x];
  meksol = FindFit[mekin, α + β * x^γ, {α, β, γ}, x];
  CPfitted[x_] = α + β * x^γ /. CPsol;
  mekfitted[x_] = α + β * x^γ /. meksol;
  p1 = Plot[CPfitted[x], {x, 2, 20}];
  p2 = ListPlot[mekin, PlotStyle → Purple];
  p3 = ListPlot[CPin];
  p4 = Plot[mekfitted[x], {x, 2, 20},
    AxesOrigin → {2, 0}, PlotStyle → Purple, PlotRange → {0, ymax}];
  Show[p4, p2, p3, p1]
 ]
```

Above module finds fits to MEK_L data and CP_L data, and combines fits with actual data into *a* plot

In[131]:=
```
GraphicsGrid[{{Plotter[CP4, mek4, 2000], Plotter[CP5, mek5, 3000]},
  {Plotter[CP6, mek6, 4000], Plotter[CP7, mek7, 5000]}}]
```

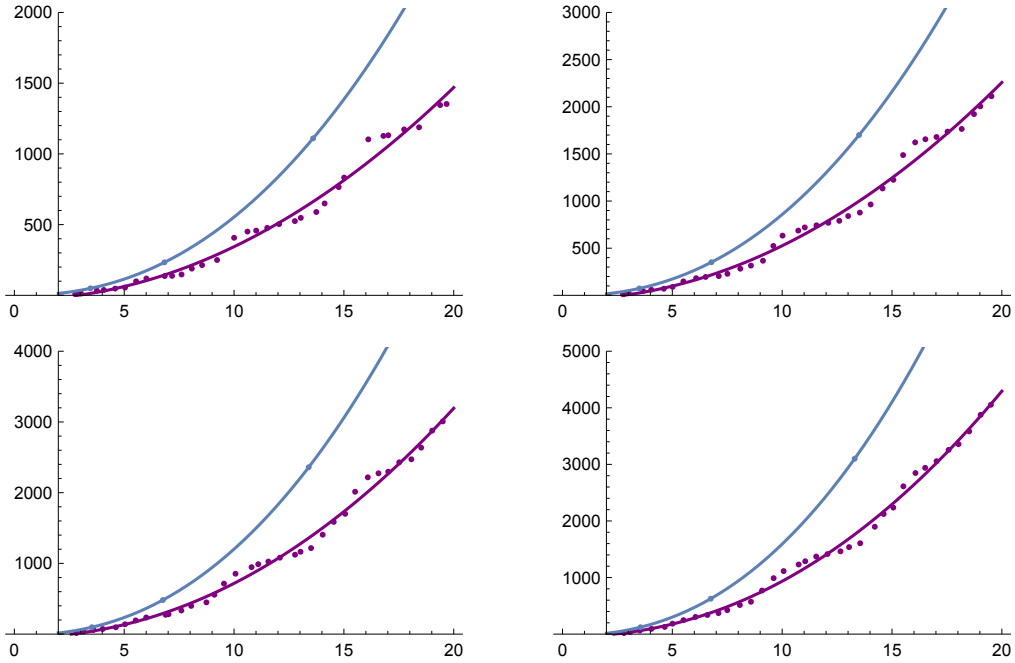
N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

General::stop : Further output of N::precsm will be suppressed during this calculation. >>

Out[131]=



Comparison of DP_L and MEK_L for various L .
Plots above used as basis for Fig 4. (b) of paper.

Gate Synthesis Cost

In[132]:=

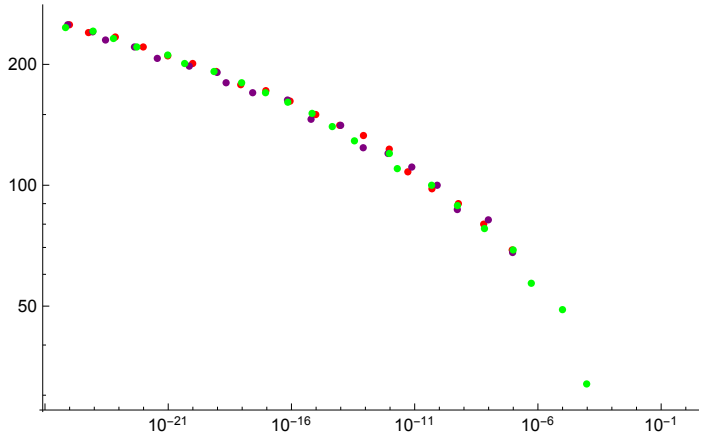
In[133]:=
```
TcountL4 = {{0.9055148479 * 10^-7, 69}, {0.6305279058 * 10^-8, 80},
  {0.5935029593 * 10^-9, 90}, {0.5016660126 * 10^-10, 98}, {0.5278333483 * 10^-11, 108},
  {0.9296883821 * 10^-12, 123}, {0.8343636825 * 10^-13, 133},
  {0.9073705356 * 10^-14, 141}, {0.9678233489 * 10^-15, 150},
  {0.8776342523 * 10^-16, 162}, {0.9148794574 * 10^-17, 172},
  {0.8757424725 * 10^-18, 178}, {0.8862784650 * 10^-19, 192},
  {0.9833541157 * 10^-20, 201}, {0.9585408890 * 10^-21, 210},
  {.9688570192 * 10^-22, 221}, {0.7129015342 * 10^-23, 234},
  {0.5825948984 * 10^-24, 240}, {0.9829428419 * 10^-25, 251}};
```

In[134]:=
```
TcountL5 = {{0.9390160772 * 10^-7, 68}, {0.9724375287 * 10^-8, 82},
    {0.5361961299 * 10^-9, 87}, {0.8184752265 * 10^-10, 100}, {0.7631573977 * 10^-11, 111},
    {0.8313466878 * 10^-12, 120}, {0.8161247655 * 10^-13, 124},
    {.9907581214 * 10^-14, 141}, {0.6245508132 * 10^-15, 146},
    {0.6913371799 * 10^-16, 163}, {0.2670556363 * 10^-17, 170},
    {0.2230139643 * 10^-18, 180}, {0.9639145159 * 10^-19, 191},
    {0.7129943030 * 10^-20, 198}, {0.3603379807 * 10^-21, 207},
    {0.4265692348 * 10^-22, 221}, {0.2860398854 * 10^-23, 230},
    {0.8594566961 * 10^-24, 241}, {0.8386713473 * 10^-25, 251}};
```

In[135]:=
```
TcountL6 = {{0.9496919476 * 10^-4, 32}, {0.9918673418 * 10^-5, 49},
    {0.5387958206 * 10^-6, 57}, {0.9907633061 * 10^-7, 69}, {0.6784972558 * 10^-8, 78},
    {0.5432027921 * 10^-9, 89}, {0.4861310976 * 10^-10, 100}, {0.1969712049 * 10^-11, 110},
    {0.9394335481 * 10^-12, 120}, {0.3596223551 * 10^-13, 129},
    {0.4527272103 * 10^-14, 140}, {0.6870568191 * 10^-15, 151},
    {0.7115712909 * 10^-16, 161}, {0.8868826632 * 10^-17, 170},
    {0.9538034702 * 10^-18, 180}, {0.7217678523 * 10^-19, 192},
    {0.4643213199 * 10^-20, 201}, {0.9559370225 * 10^-21, 211},
    {0.5313943784 * 10^-22, 221}, {0.6004551666 * 10^-23, 232},
    {0.8932720183 * 10^-24, 242}, {0.6824269773 * 10^-25, 247}};
ListLogLogPlot[{TcountL4, TcountL5, TcountL6}, PlotStyle → {Red, Purple, Green}]
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[136]=



Above lists give the error rate and T-count of performing gates in the L=4,5,6 levels of the Clifford hierarchy. All obtained from gridsynth http://www.mathstat.dal.ca/~selinger/newsynth/
We also plot this data

In[137]:=
```
SizeOfLevel3Cost = Dimensions[Level3Cost][[1]];

FindCombinedCost[Tcount_] := Module[{SizeOfTcount, a, b, output},
   SizeOfTcount = Dimensions[Tcount][[1]];
   output = {{ϵin, 1}};
   For[a = 0, a < SizeOfLevel3Cost, a++;
    For[b = 0, b <  SizeOfTcount, b++;
     ϵtotal = Tcount[[b, 1]] + Tcount[[b, 2]] * Level3Cost[[a, 1]];
     Cost = Tcount[[b, 2]] * Level3Cost[[a, 2]];
     output = Append[output, {ϵtotal, Cost}];
    ];
   ];
   BestOf[output]
  ];
```

In[139]:=
```
TotalCost4 = FindCombinedCost[TcountL4];
TotalCost5 = FindCombinedCost[TcountL5];
TotalCost6 = FindCombinedCost[TcountL6];
```

In[142]:=

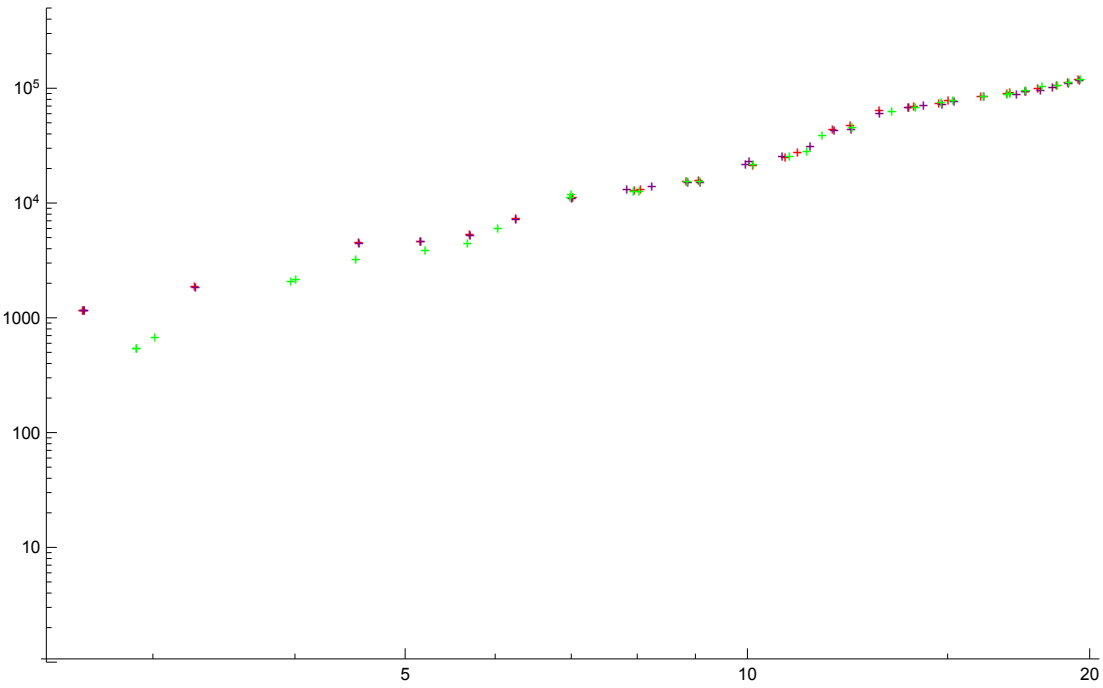
In[143]:=
```
G4 = ListLogLogPlot[MakePlotList[TotalCost4],
   PlotStyle → Red, PlotMarkers → {"+"}, PlotRange → {1, 500 000}];
G5 = ListLogLogPlot[MakePlotList[TotalCost5], PlotStyle → Purple,
   PlotMarkers → {"+"}, PlotRange → {1, 500 000}];
G6 = ListLogLogPlot[MakePlotList[TotalCost6], PlotStyle → Green,
   PlotMarkers → {"+"}, PlotRange → {1, 500 000}];
Gall = Show[G4, G5, G6]
```

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

N::precsm : Requested precision 16 is smaller than $MinPrecision. Using $MinPrecision instead. >>

Out[146]=

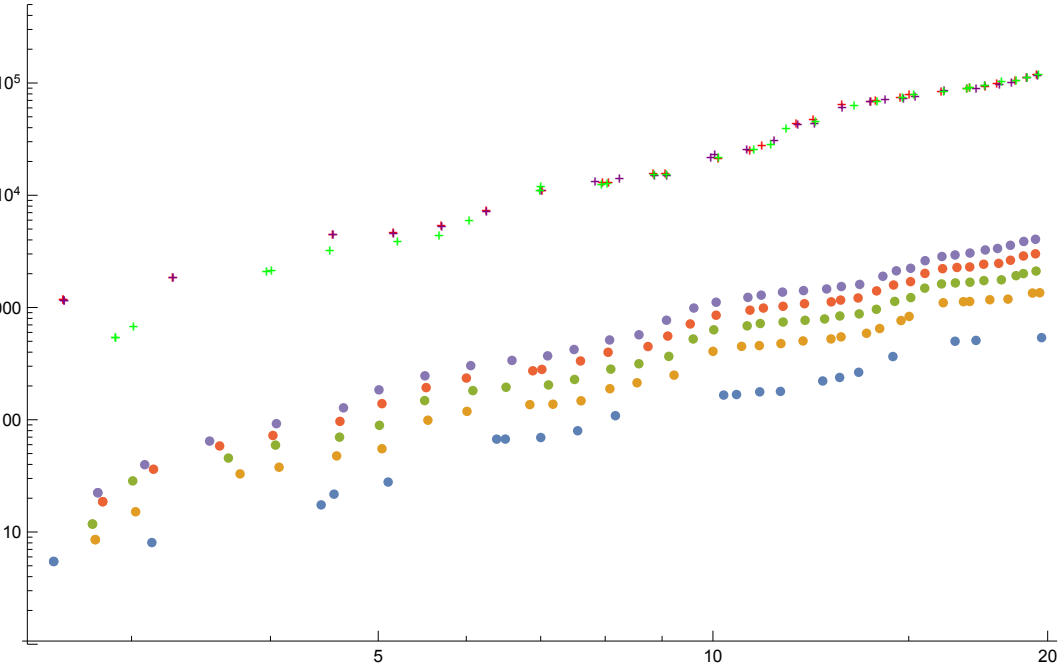


In[147]:=

Combined Tcount and magic state cost for L=4,5,6.

In[148]:=
```
Show[Gall, mekALL]
```

Out[148]=



The above compares the gate synthesis data with the MEK_L data.

Plots above used as basis of Fig. 5 of paper.