IIR Filter design

14.1 Ustalenie postaci filtrów cyfrowych

Poniżej znajdują się definicje filtrów analogowych i ich cyfrowych odpowiedników w postaci tabel dla rzędu N={1,2,...7} oraz częstości ω={ωmin, ωmax, $\Delta\omega$}

14.1.1 Definicje

In[290]:=
```
Δω = 0.3;
ωmin = 0.1;
ωmax = 2.0;
```

14.1.2 Butterworth

```
In[293]:= bmodels = Table[TransferFunctionFactor@ButterworthFilterModel[{"Lowpass", n, ω}, s],
            {n, 1, 7, 1}, {ω, ωmin, ωmax, Δω}];
          DGbmodels =
            Table[TransferFunctionFactor@
              ToDiscreteTimeModel[ButterworthFilterModel[{"Lowpass", n, ω}, s], 1], {n, 1, 7, 1},
             {ω, ωmin, ωmax, Δω}];
```

14.1.3 Chebyshev 1

```
In[295]:= c1models = Table[TransferFunctionFactor@Chebyshev1FilterModel[{"Lowpass", n, ω}, s],
            {n, 1, 7, 1}, {ω, ωmin, ωmax, Δω}];
          DGc1models =
            Table[TransferFunctionFactor@ToDiscreteTimeModel[Chebyshev1FilterModel[{"Lowpass", n, ω}, s],
               1], {n, 1, 7, 1}, {ω, ωmin, ωmax, Δω}];
```

14.1.4 Chebyshev 2

```
In[297]:= c2models = Table[TransferFunctionFactor@Chebyshev2FilterModel[{n, ω}, s], {n, 1, 7, 1},
            {ω, ωmin, ωmax, Δω}];
          DGc2models =
            Table[TransferFunctionFactor@ToDiscreteTimeModel[Chebyshev2FilterModel[{n, ω}, s], 1],
             {n, 1, 7, 1}, {ω, ωmin, ωmax, Δω}];
```

14.1.5 Eliptic

```
In[299]:= emodels = Table[TransferFunctionFactor@EllipticFilterModel[{n, ω}, s], {n, 1, 7, 1},
            {ω, ωmin, ωmax, Δω}];
          DGemodels = Table[TransferFunctionFactor@ToDiscreteTimeModel[EllipticFilterModel[{n, ω}, s], 1],
            {n, 1, 7, 1}, {ω, ωmin, ωmax, Δω}];
```

14.2 Dyskretyzacja na poziomie zer i biegunów

14.2.1 Wstęp

Na początek przedstawiam sposób w jaki dyskretyzowałem dotychczas, tj. mając filtr cyfrowy w postaci $\frac{a_0(z-a_1)}{(z-b_1)\,(z-b_2)}$ dyskretyzowałem liczby $a_1,\ b_1,\ b_2$.

Okazuje się, że nie jest najlepszy sposób dyskretyzacji by sprawdzić rzeczywiste efekty tego procesu.

Lepszy sposób prezentowany w kolejnym paragrafie, polega na dyskretyzacji końcowych współczynników filtru, a więc współczynników $c_0,\ c_1,\ d_0,\ d_1,\ d_2$ filtru $\frac{c_0 + c_1\,z}{d_0 + d_1\,z + d_2\,z^2}$ będącego wymnożoną postacią wyżej przedstawionego filtru.

14.2.2 Definicje

```
In[3]:= DiscretizeList[x_, max_, bits_] :=
          If[# ≥ 0 , 1, -1] Round[(Abs[#] / max) (Power[2, bits - 1] - 1)] max / (Power[2, bits - 1] - 1) & /@ x;
```

```
In[279]:= DiscretizeComplexLists[x_, y_, coeff_, bits_] := Module[{xD, yD, coeffD, max},
            If[y == {}, {},
             max = Max[Abs[Join[Re@x, Im@x, Re@y, Im@y, Re@{coeff}, Im@{coeff}]]];
             xD = DiscretizeList[Re@x, max, bits] + i DiscretizeList[Im@x, max, bits];
             yD = DiscretizeList[Re@y, max, bits] + i DiscretizeList[Im@y, max, bits];
             coeffD = Total[DiscretizeList[{Re@coeff, i Im@coeff}, max, bits]];
             {xD, yD, coeffD}]
            ];
          DiscretizeModel[tf_, bits_] := Module[{tfD, coeff, zeros, poles, coeffD, zerosD, polesD},
            coeff = First[Flatten[Numerator[tf[z]]]];
            coeff = If[Length[coeff] > 0, coeff[[1]], coeff];
            zeros = (z /. Solve[Numerator@(tf[z]) == 0, z]) /. {z → {}};
            poles = (z /. Solve[Denominator@(tf[z] / coeff) == 0, z]);
            {zerosD, polesD, coeffD} = DiscretizeComplexLists[zeros, poles, Last@poles, bits];
            (*nie wykorzystuję coeffD, bo często jest bardzo mały i przechodzi w 0 podczas
             dyskretyzacji, by nie zaburzyć wyniku, podają mu inny, istniejący biegun*)
            Chop[coeff (z - zerosD /. {List → Times}) / (z - polesD /. {List → Times})]
            ];
```

Pomocnicza funkcja by pobrać bieguny zdyskretyzowanych filtrów

```
In[53]:= ExtractPoles[tf_] := Module[{coeff},
           coeff = First[Flatten[Numerator[tf]]];
           coeff = If[Length[coeff] > 0, coeff[[1]], coeff];
           (z /. Solve[Denominator@(tf / coeff) == 0, z])
          ]
```

Ustalamy ilość bitów do symulacji

```
In[365]:= bity = 6;
```

14.2.3 Testy

```
In[54]:= {a, b, c} = DiscretizeComplexLists[{1, 2, 3}, {i, i 2, i 3}, 2, 6]
```

$$\text{Out[54]}= \left\{\left\{\frac{30}{31}, \frac{63}{31}, 3\right\}, \left\{\frac{30\,i}{31}, \frac{63\,i}{31}, 3\,i\right\}, \frac{63}{31}\right\}$$

Dyskretyzujemy na poziomie filtrów cyfrowych

```
In[13]:= buttModel = DGbmodels[[1, 1]]
```

$$\text{Out[13]}= \left(\frac{0.047619\ (1. + \dot{z})}{-0.904762 + \dot{z}}\right)_{\mathcal{T}}$$

```
In[61]:= polesButt = TransferFunctionPoles[buttModel][[1, 1]]
Out[61]= {0.904762}

In[231]:= buttModelD = DiscretizeModel[buttModel, bity]
Out[231]= 0.047619 (1. + z) / (-0.903226 + z)

In[60]:= polesButtD = ExtractPoles[buttModelD]
Out[60]= {0.90411}
```

Ale funkcja dyskretyzująca działa też dla filtrów analogowych

```
In[15]:= buttModel2 = bmodels[[3, 3]]
Out[15]= ( 0.125 / ((0.25 - 0.433013 i) + s) ((0.25 + 0.433013 i) + s) (0.5 + s) )^T

In[281]:= buttModel2D = DiscretizeModel[buttModel2, bity]
Out[281]= 0.125 / ((0.258065 - 0.435484 i) + z) ((0.258065 + 0.435484 i) + z) (0.5 + z)
```

14.2.4 Dyskretyzacja

```
In[301]:= DGbmodelsD = Map[DiscretizeModel[#, bity] &, DGbmodels, {2}];

In[302]:= DGc1modelsD = Map[DiscretizeModel[#, bity] &, DGc1models, {2}];

In[303]:= DGc2modelsD = Map[DiscretizeModel[#, bity] &, DGc2models, {2}];

In[304]:= DGemodelsD = Map[DiscretizeModel[#, bity] &, DGemodels, {2}];
```

14.2.5 Porównanie położenia biegunów

★ Definicje

Przygotowywuję funkcję rysującą bieguny i zaznaczającą na czerwono te w których bieguny filtru zdyskretyzowanego wychodzą poza okrąg jednostkowy.

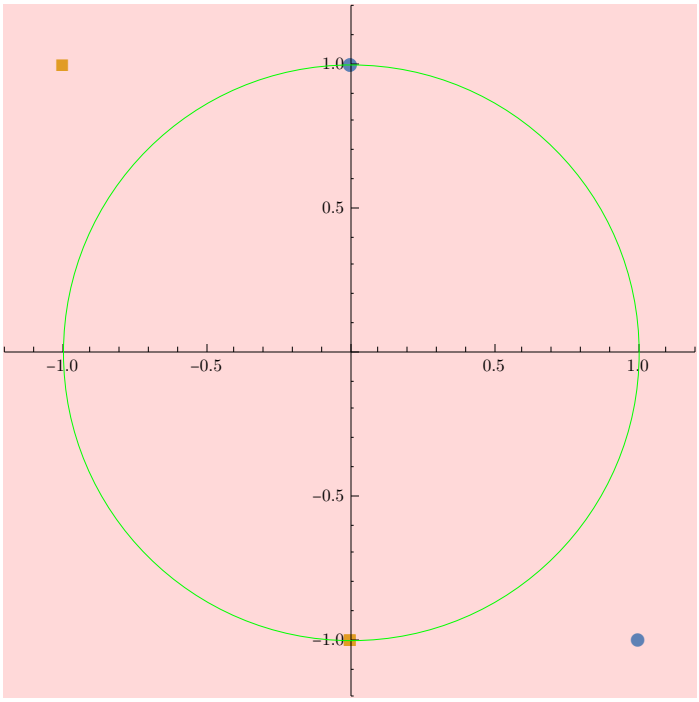
In[305]:=
```
PlotPoles[poles_, poles2_] :=
 ListPlot[{Transpose[{Re[poles], Im[poles]}], Transpose[{Re[poles2], Im[poles2]}]},
  PlotRange → 1.2 {{-1, 1}, {-1, 1}}, PlotStyle → {PointSize[Large]},
  Background → If[Count[poles2, _?(Abs[#] ≥ 1 &), {1}] > 0, LightRed, White],
  PlotMarkers → {Automatic, Medium}, AxesOrigin → {0, 0}, Epilog → {Green, Circle[]},
  AspectRatio → Automatic]

AsymptoticallyStableQ[tfm_?ContinuousTimeModelQ] := If[
  Count[TransferFunctionPoles[tfm], Complex[x_?NonNegative, _] | x_?NonNegative, {3}] > 0,
  False, True]
AsymptoticallyStableQ[tfm_?DiscreteTimeModelQ] := If[
  Count[TransferFunctionPoles[tfm], _?(Abs[#] ≥ 1 &), {3}] > 0, False, True]
```

★ Testy

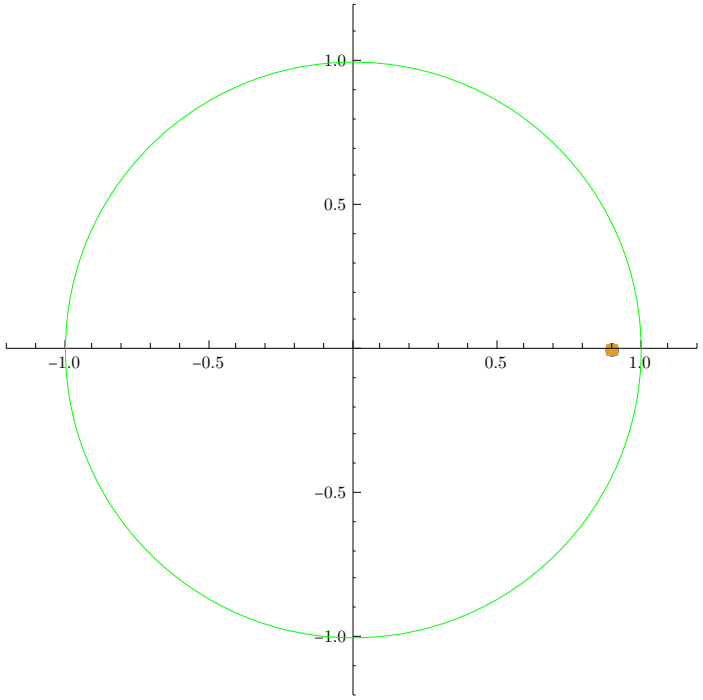
In[265]:=
```
PlotPoles[{1 - i, i}, -{1 - i, i}]
```

Out[265]=



In[264]:=
```
PlotPoles[polesButt, polesButtD]
```

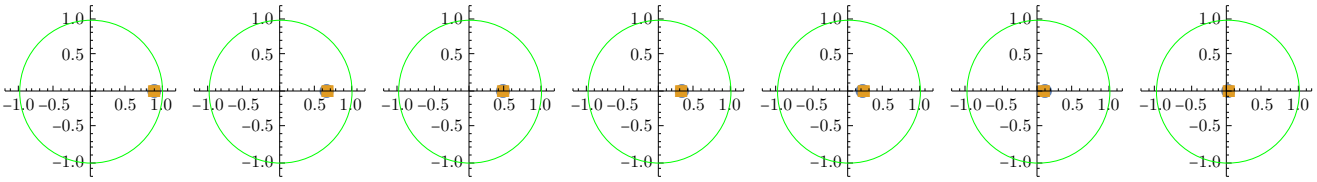
Out[264]=



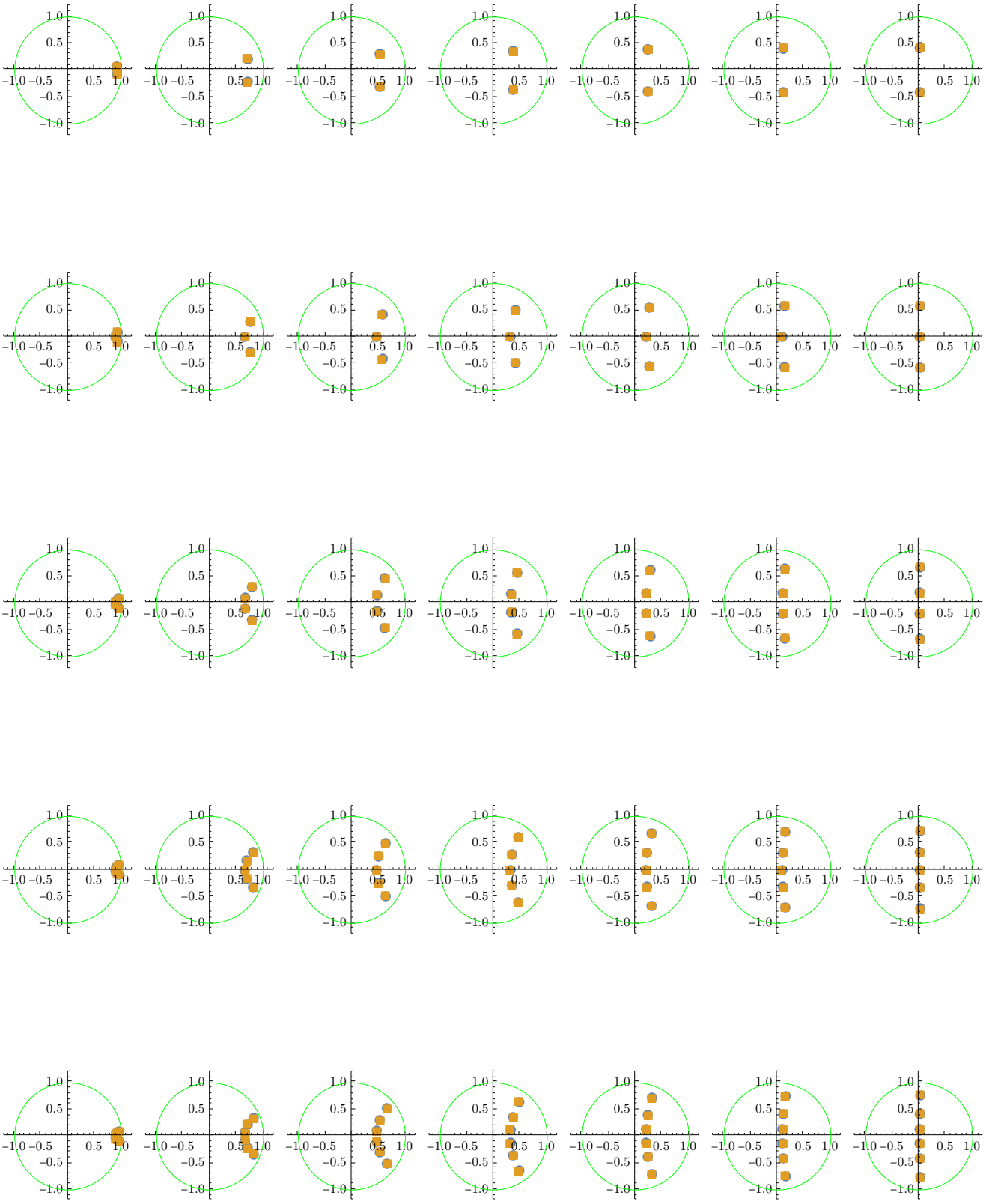
★ Butterworth

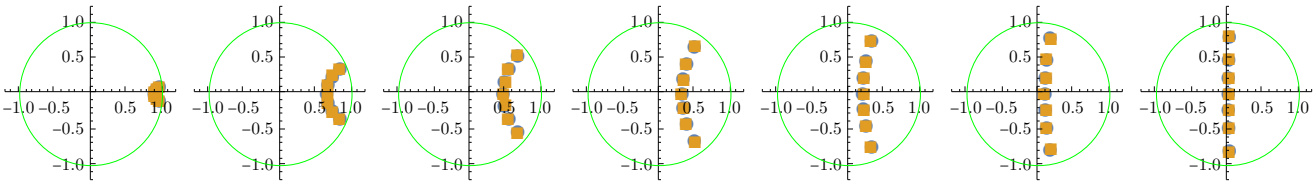
W prawo rośnie częstotliwość, w dół rośnie rząd filtra.

In[308]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGbmodels, DGbmodelsD}, 2]
```



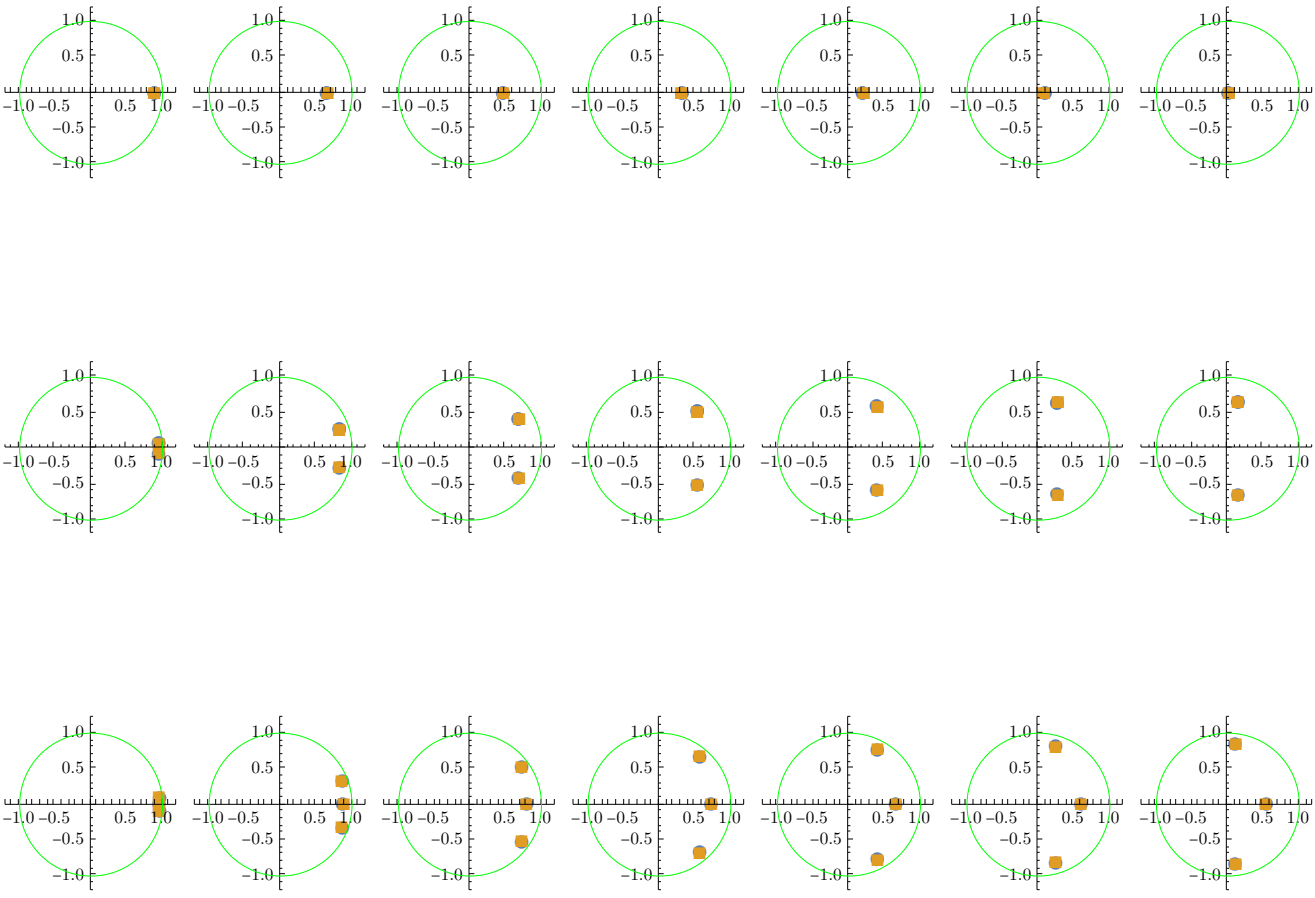
Out[308]=



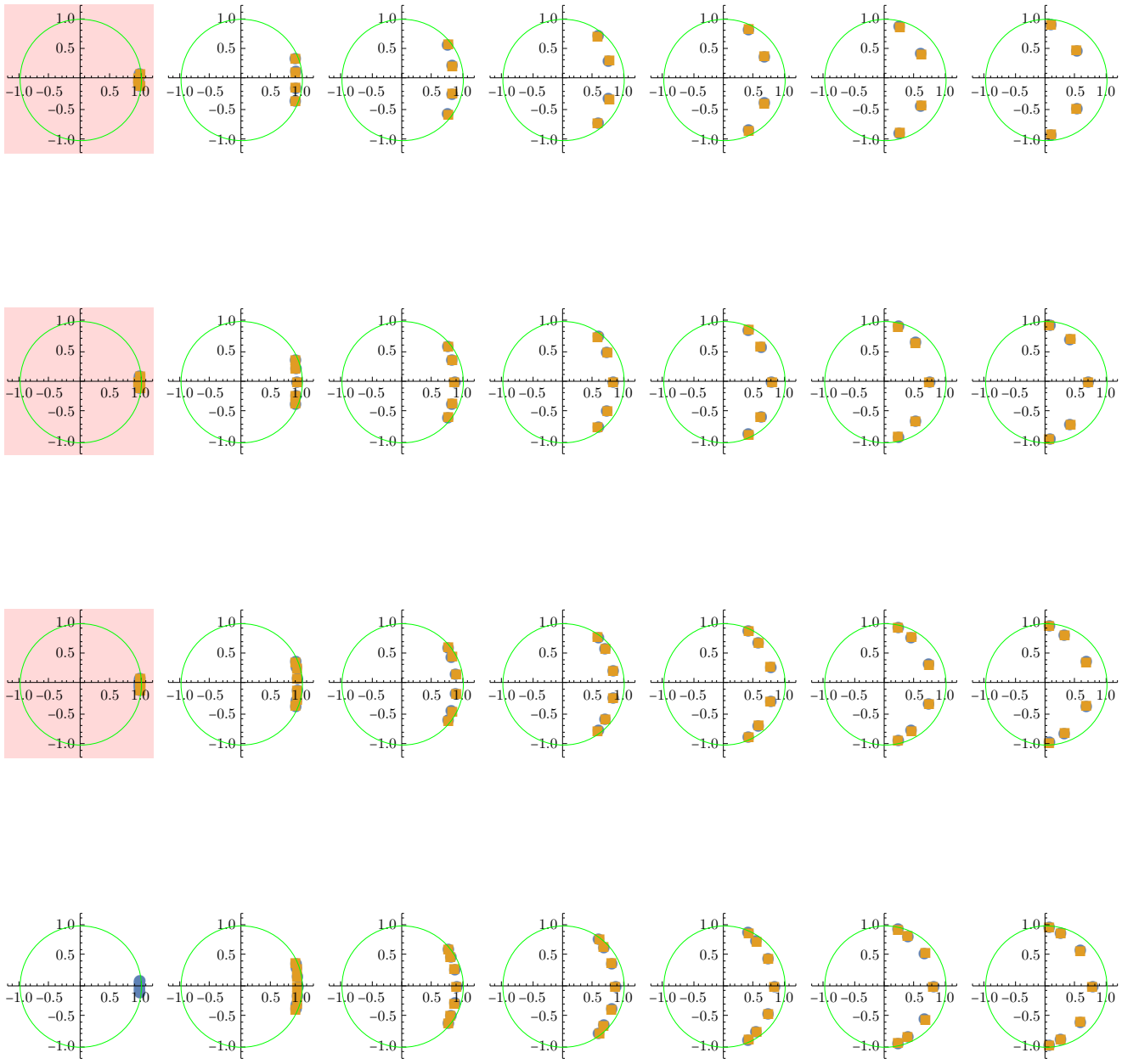


★ Chebyshev 1

In[309]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGc1models, DGc1modelsD}, 2]
```



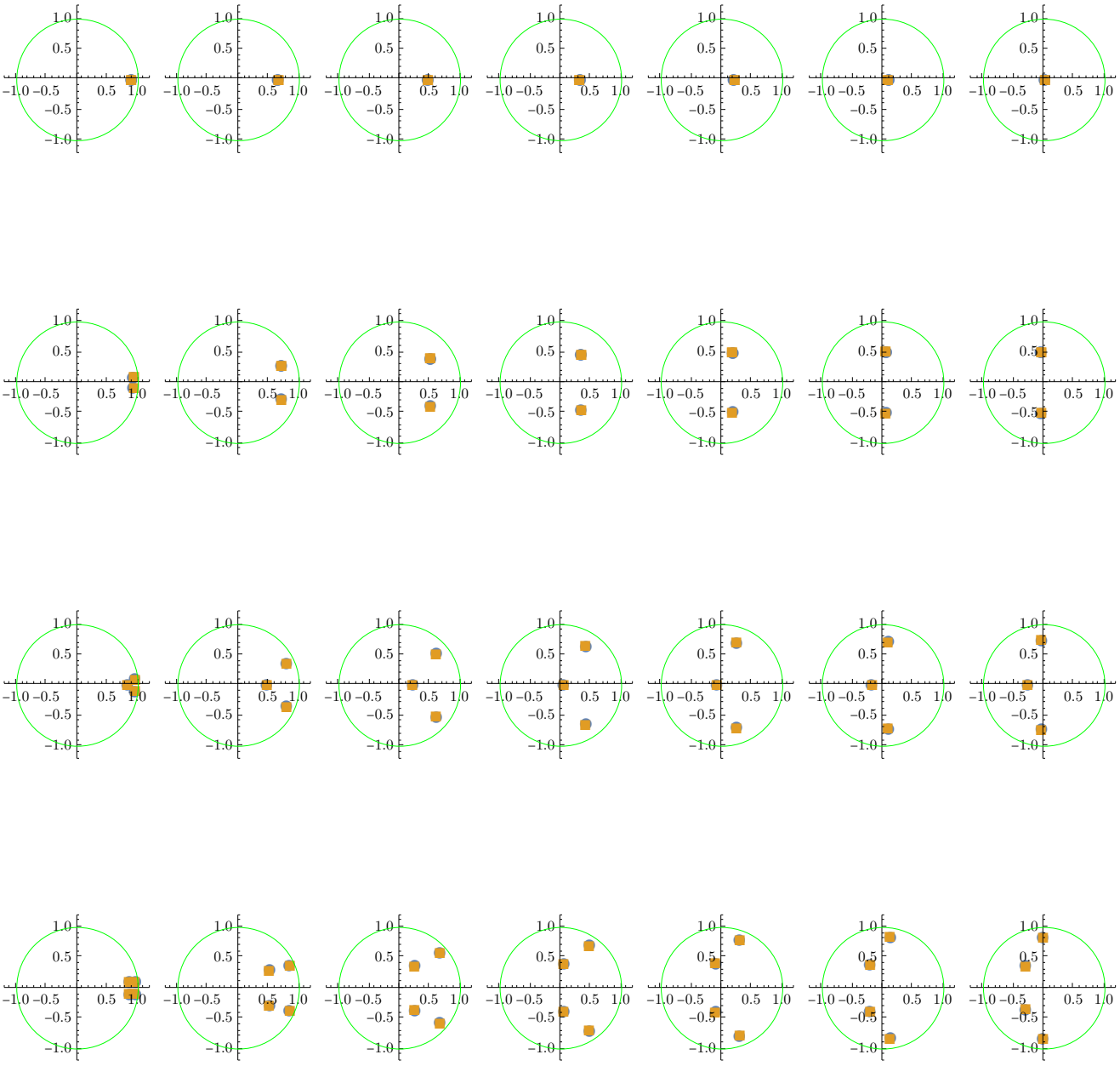
Out[309]=

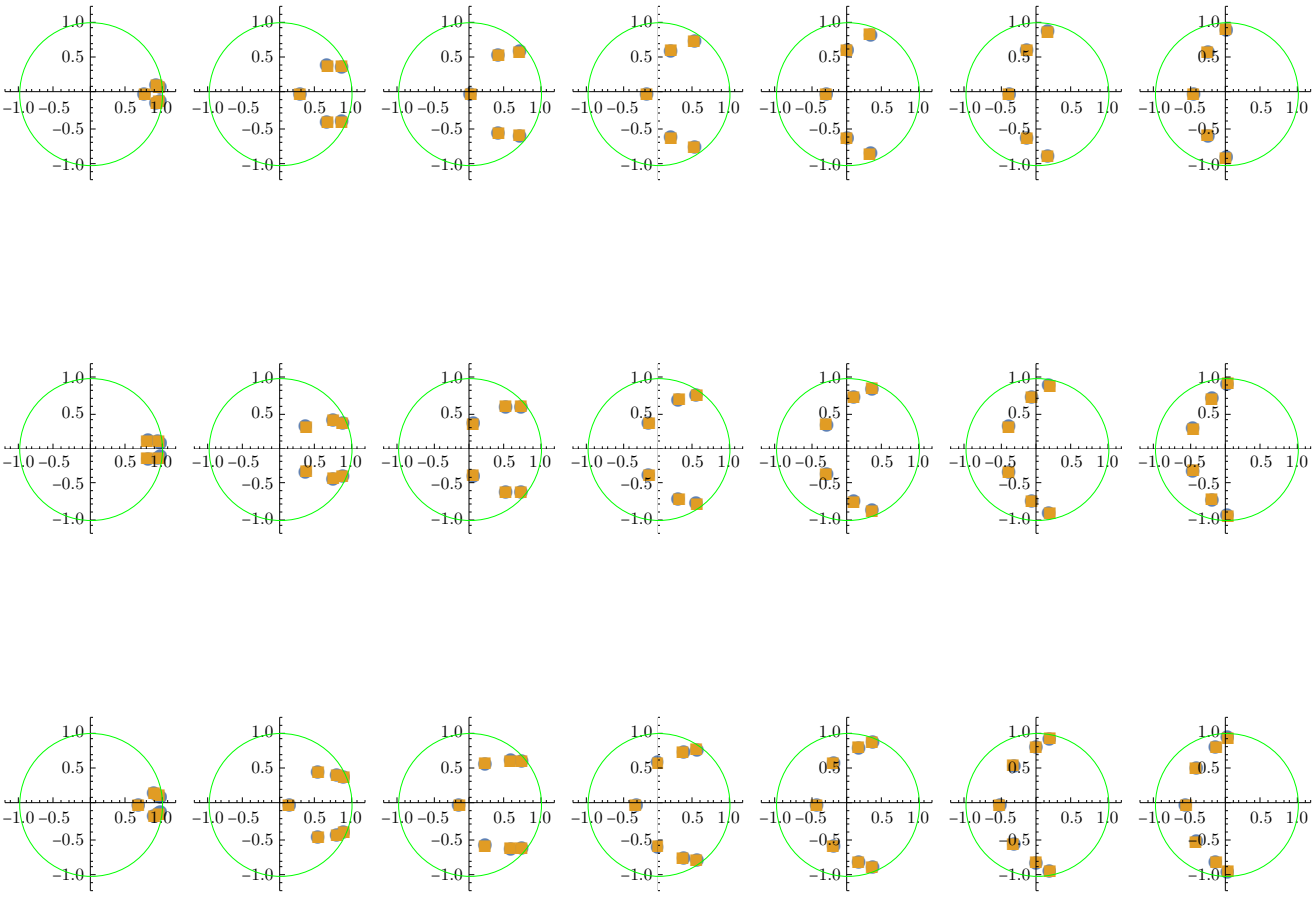


★ Chebyshev 2

In[310]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGc2models, DGc2modelsD}, 2]
```

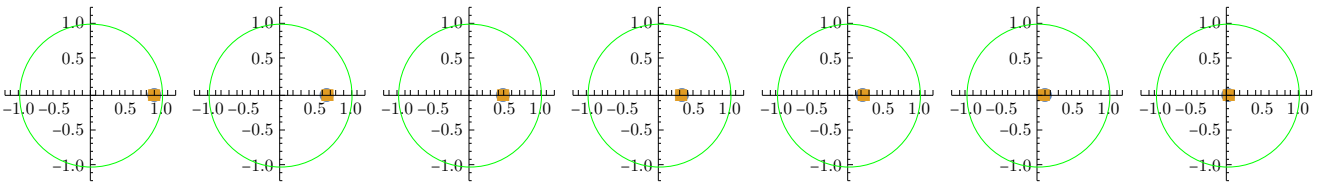
Out[310]=



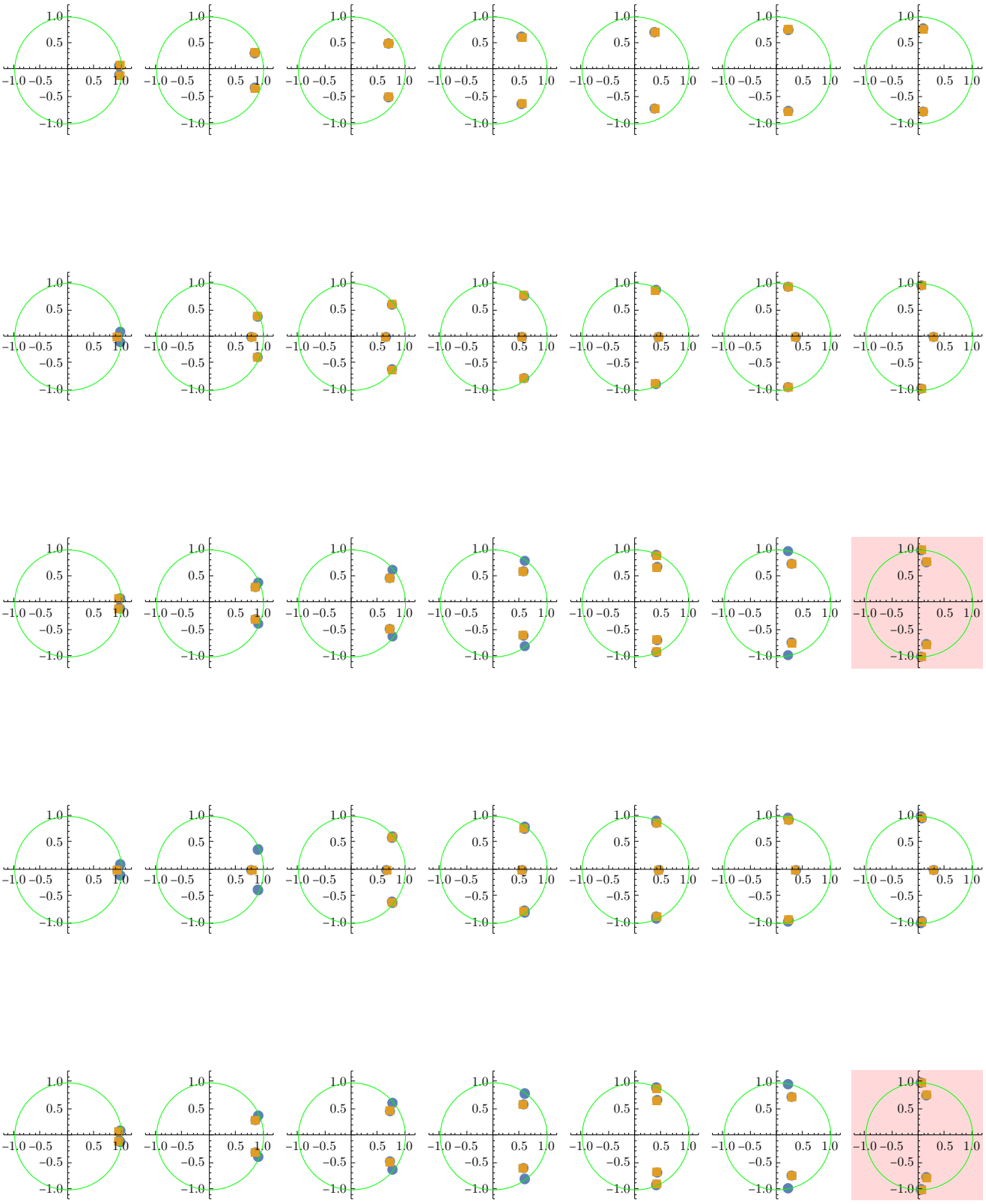


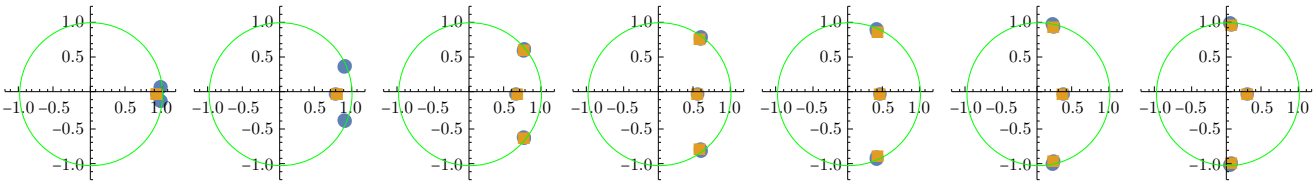
★ Eliptyczne

In[311]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGemodels, DGemodelsD}, 2]
```



Out[311]=





14.3 Dyskretyzacja na poziomie współczynników

14.3.1 Definicje

In[359]:=
```
DiscretizeList[x_, max_, bits_] :=
  If[# ≥ 0 , 1, -1] Round[(Abs[#] / max) (Power[2, bits - 1] - 1)] max / (Power[2, bits - 1] - 1) & /@ x;
DiscretizeComplexList[x_, y_, bits_] := Module[{xD, yD, coeffD, max},
   If[y == {}, {},
    max = Max[Abs[Join[Re@x, Im@x, Re@y, Im@y]]];
    xD = DiscretizeList[Re@x, max, bits] + i DiscretizeList[Im@x, max, bits];
    yD = DiscretizeList[Re@y, max, bits] + i DiscretizeList[Im@y, max, bits];
    {xD, yD}]
  ];
DiscretizeModelCoeffs[tf_, bits_] := Module[{zeros, poles, zerosD, polesD},
       zeros = CoefficientList[Numerator[TransferFunctionExpand[tf][z]][[1, 1]], z];
   poles = CoefficientList[Denominator[TransferFunctionExpand[tf][z]][[1, 1]], z];
   {zerosD, polesD} = DiscretizeComplexList[zeros, poles, bits];
       Chop[FromDigits[Reverse[zerosD], z] / (FromDigits[Reverse[polesD], z])]
  ];
```

14.3.2 Testy

In[363]:=
```
{a, c} = DiscretizeComplexList[{1, 2, 3}, {i, i 2, i 3}, 2]
```

Out[363]=
$$\{\{0, 3, 3\}, \{0, 3\,i, 3\,i\}\}$$

Dyskretyzujemy na poziomie filtrów cyfrowych

In[371]:=
```
buttModel = TransferFunctionExpand@DGbmodels[[2, 2]]
```

Out[371]=
$$\left(\frac{0.0302379 + 0.0604758\,\dot{z} + 0.0302379\,\dot{z}^2}{(0.572371 + 0.\,i) - (1.45142 + 0.\,i)\,\dot{z} + \dot{z}^2}\right)_{\mathcal{T}}$$

In[373]:=
```
polesButt = TransferFunctionPoles[buttModel][[1, 1]]
```

Out[373]=
$$\{0.72571 - 0.213814\,i, 0.72571 + 0.213814\,i\}$$

In[374]:=
```
buttModelD = N@DiscretizeModelCoeffs[buttModel, bity]
```

Out[374]=
$$\frac{0.04682 + 0.04682\,z + 0.04682\,z^2}{0.56184 - 1.45142\,z + 0.98322\,z^2}$$

In[375]:=
```
polesButtD = ExtractPoles[buttModelD]
```

Out[375]=
$$\{0.738095 - 0.16323\,i,\ 0.738095 + 0.16323\,i\}$$

14.3.3 Dyskretyzacja

In[376]:=
```
DGbmodelsDc = Map[DiscretizeModelCoeffs[#, bity] &, DGbmodels, {2}];
```

In[377]:=
```
DGc1modelsDc = Map[DiscretizeModelCoeffs[#, bity] &, DGc1models, {2}];
```

In[378]:=
```
DGc2modelsDc = Map[DiscretizeModelCoeffs[#, bity] &, DGc2models, {2}];
```

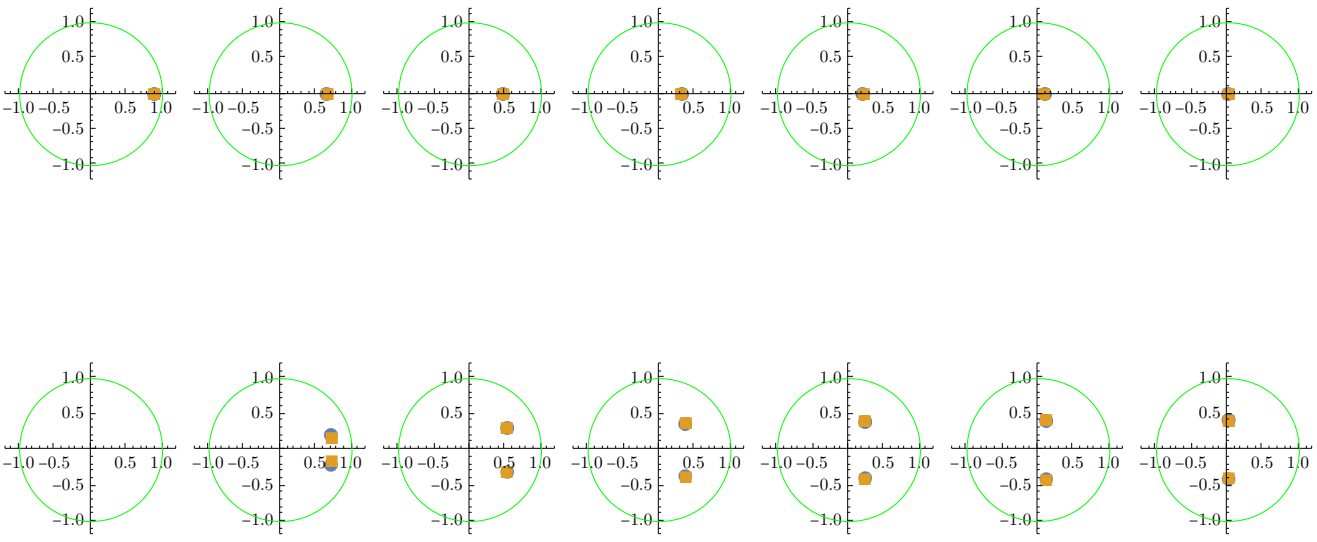
In[379]:=
```
DGemodelsDc = Map[DiscretizeModelCoeffs[#, bity] &, DGemodels, {2}];
```

14.3.4 Porównanie położenia biegunów

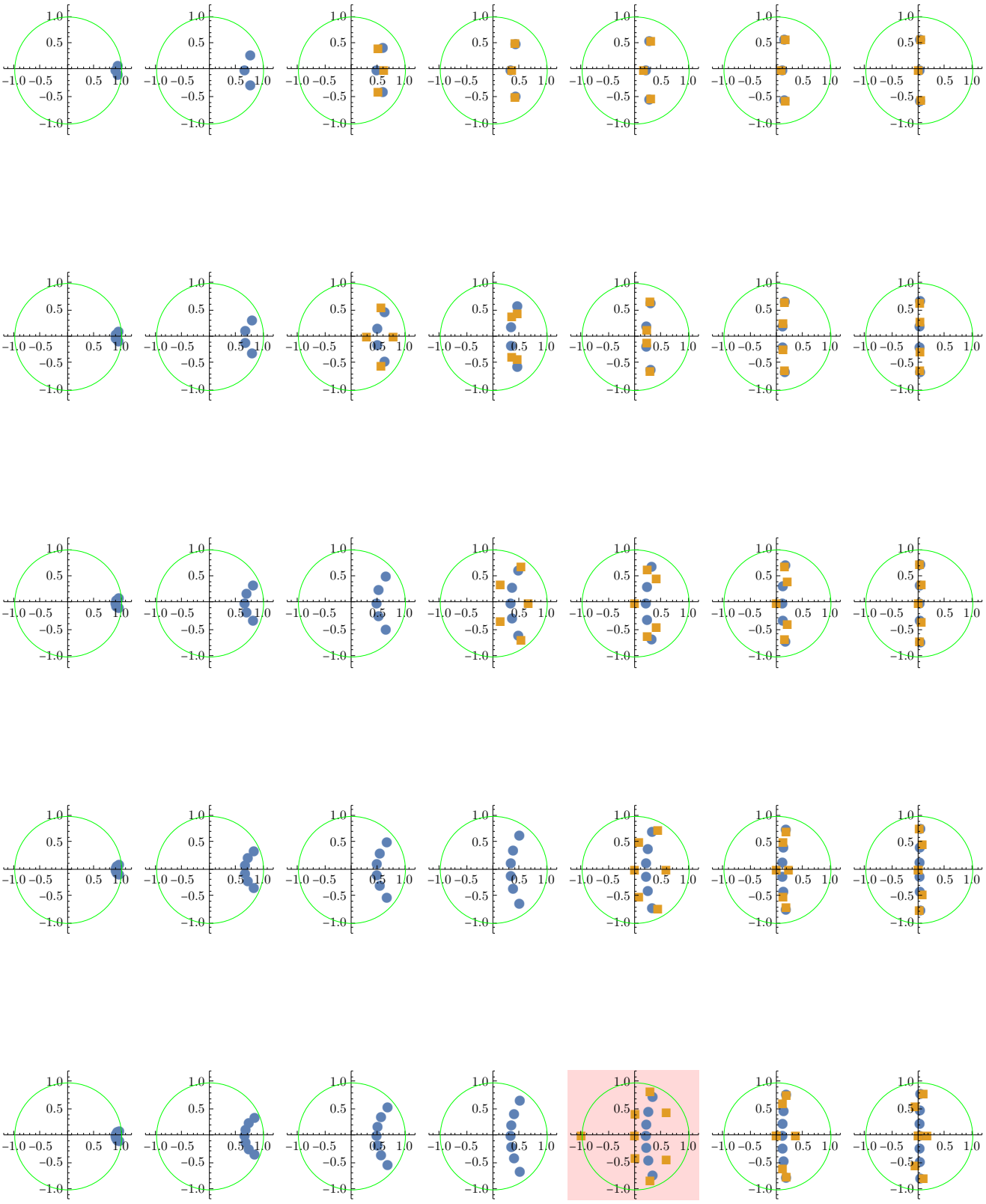
★ Butterworth

W prawo rośnie częstotliwość, w dół rośnie rząd filtra.

In[380]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGbmodels, DGbmodelsDc}, 2]
```



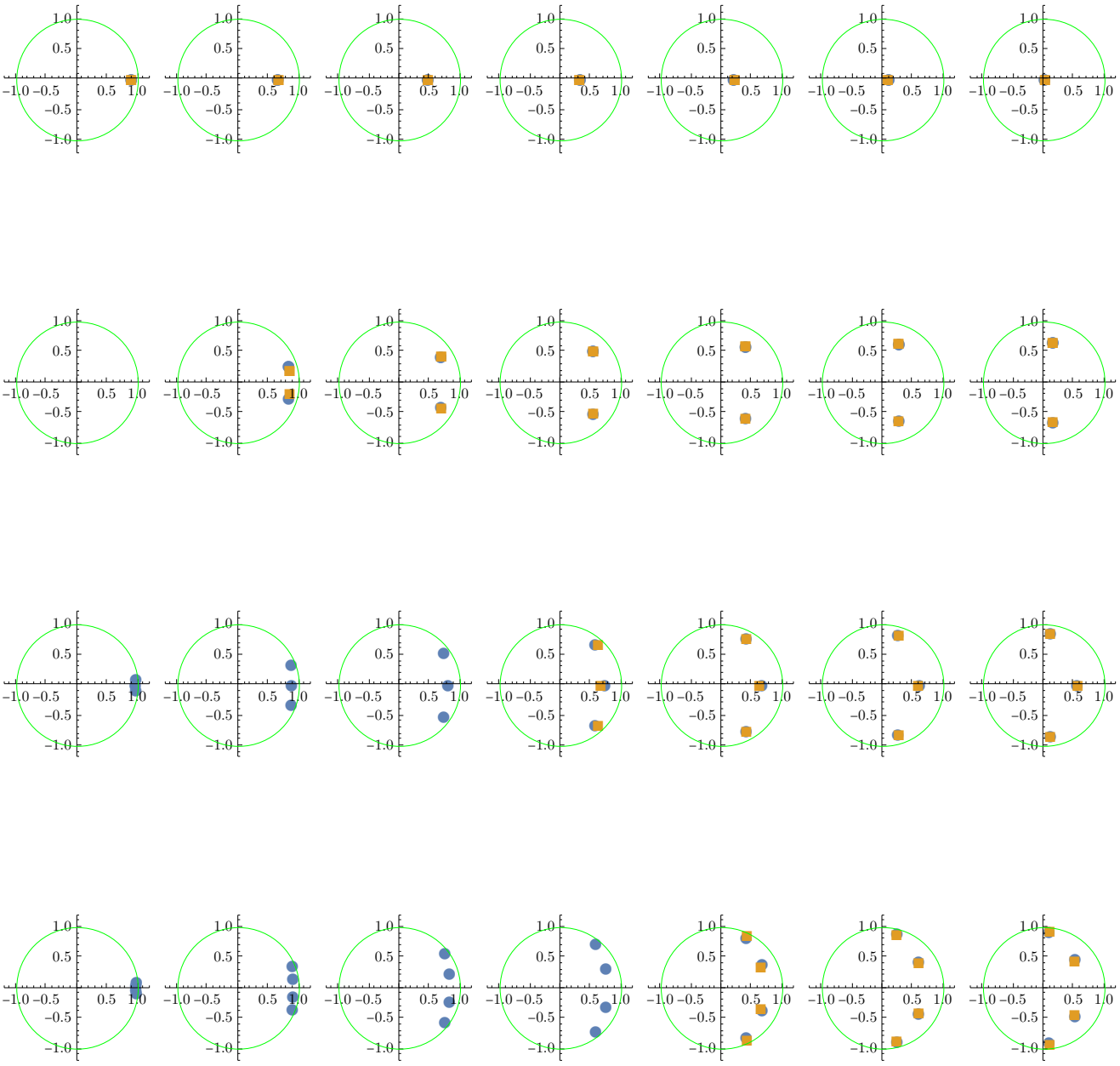
Out[380]=

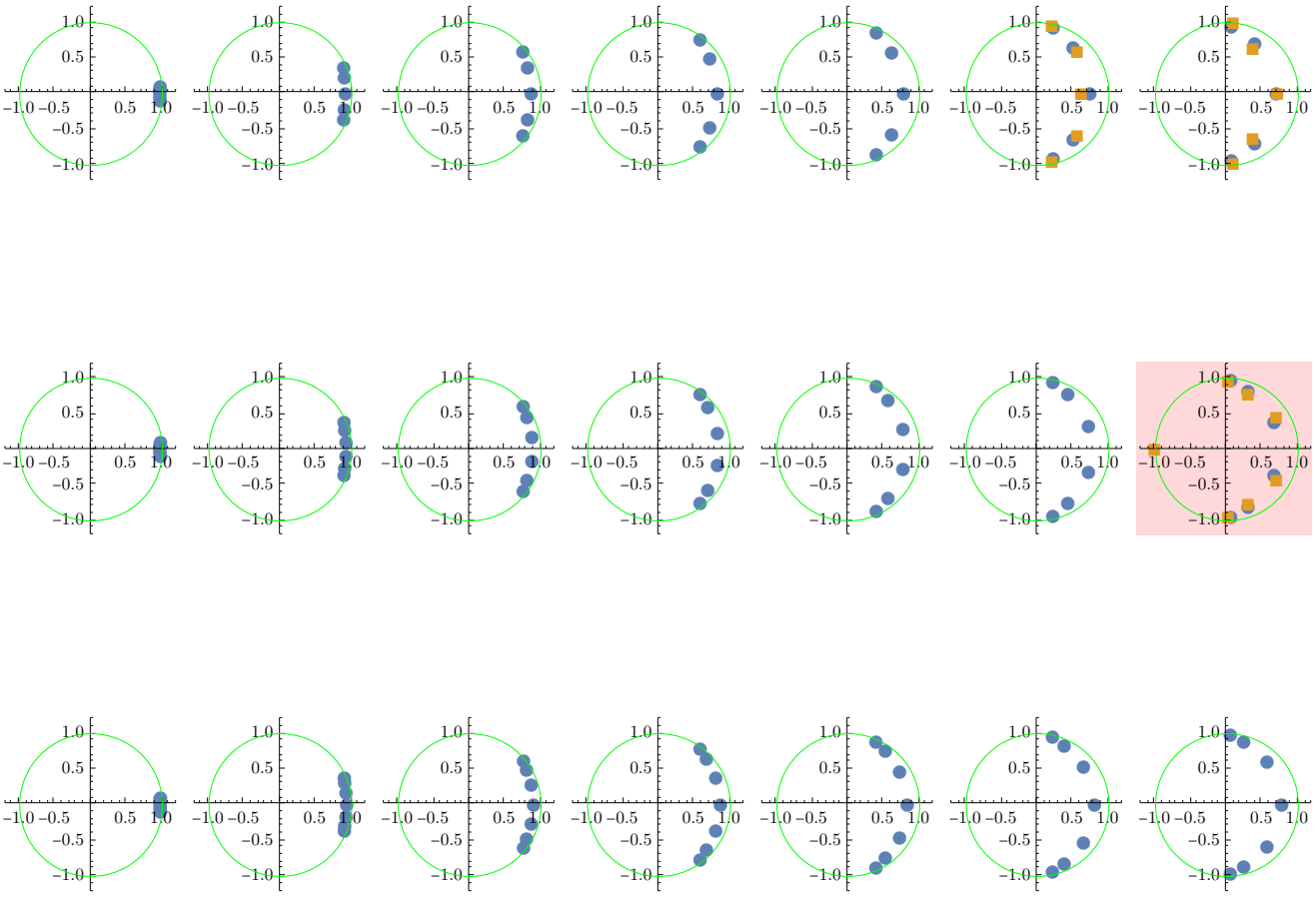


★ Chebyshev 1

In[381]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGc1models, DGc1modelsDc}, 2]
```

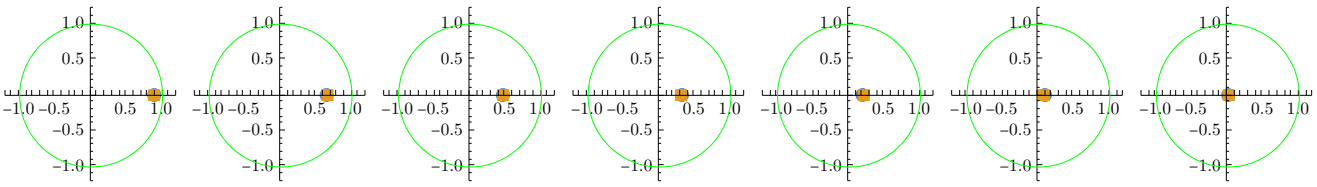
Out[381]=



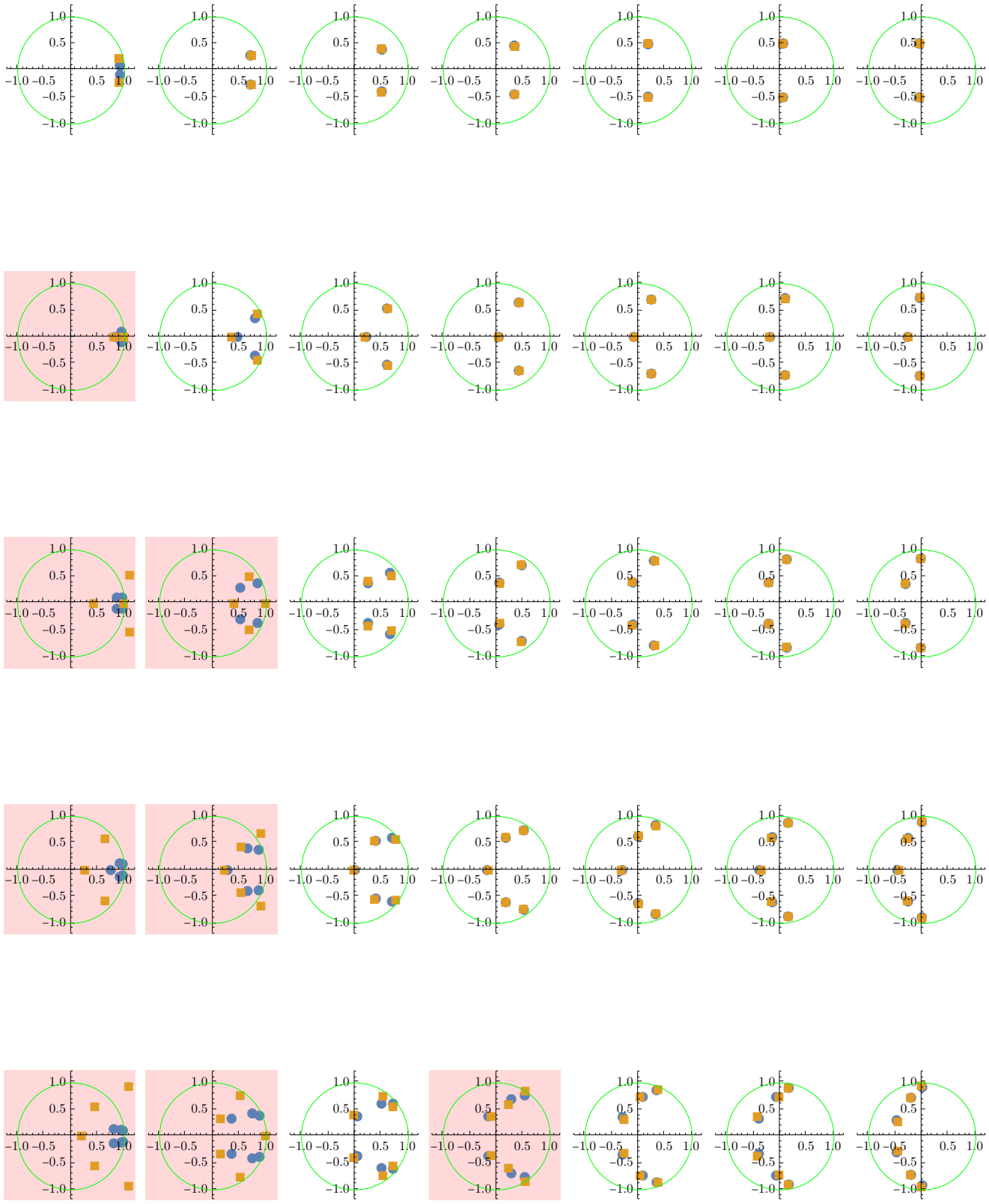


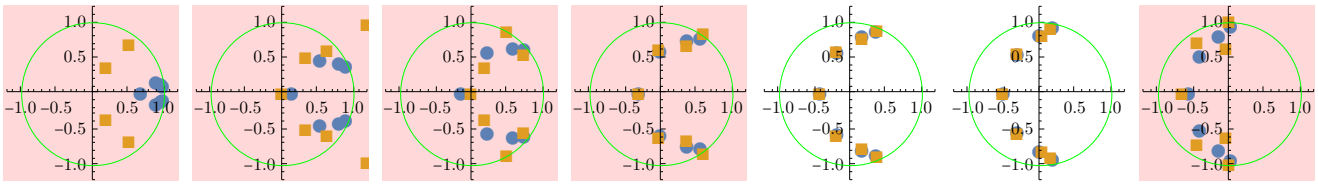
★ Chebyshev 2

In[382]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGc2models, DGc2modelsDc}, 2]
```



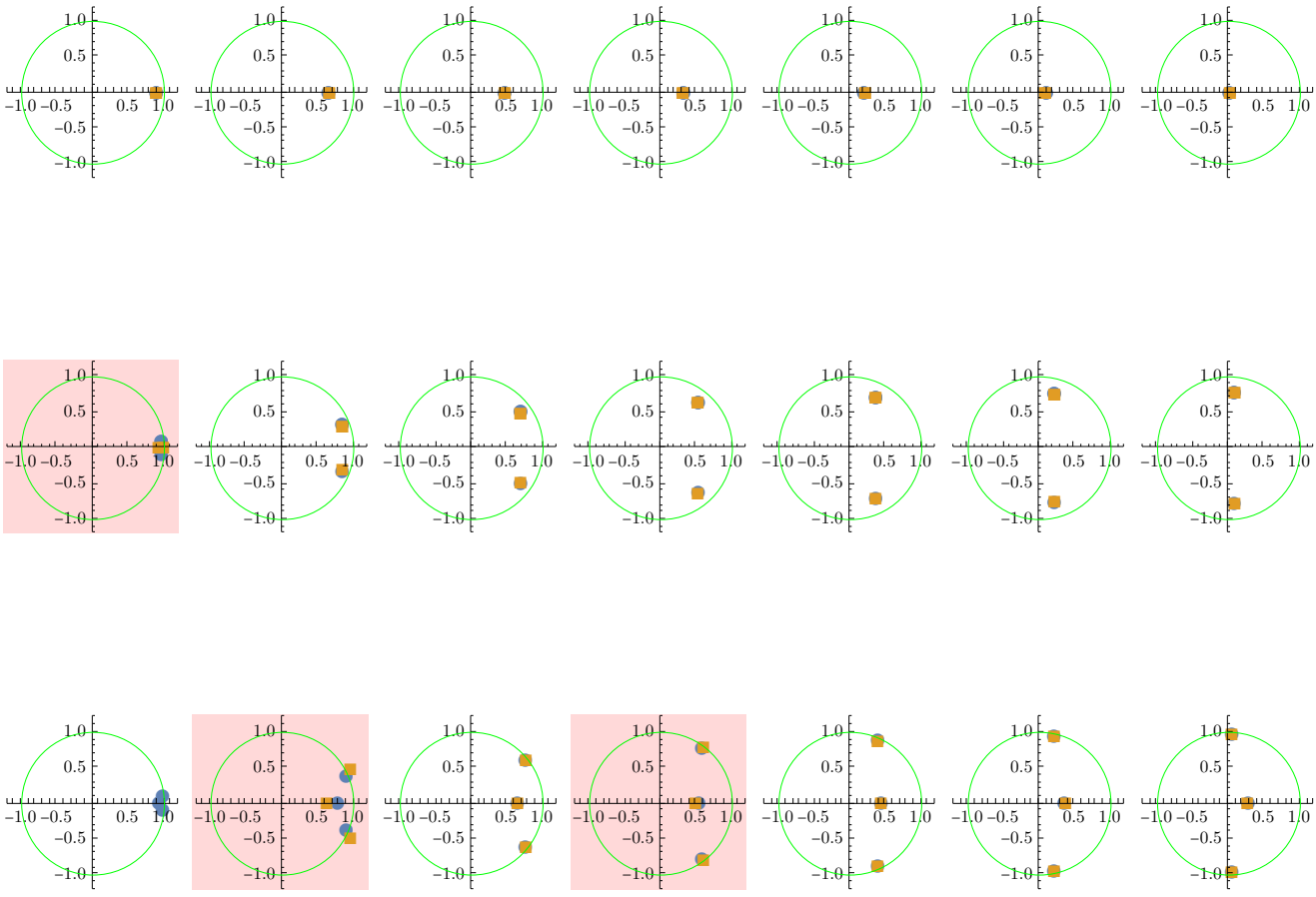
Out[382]=



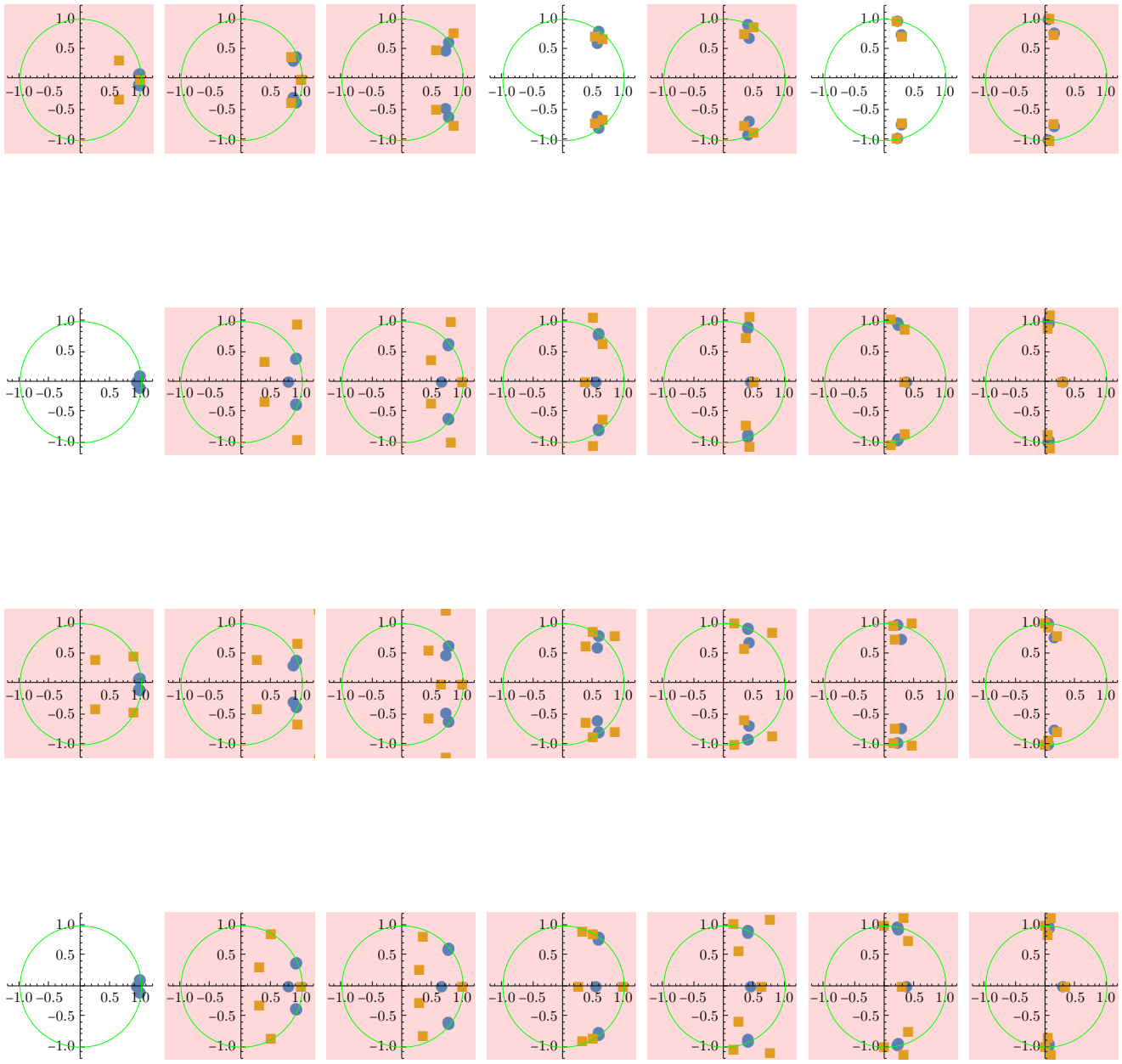


★ Eliptyczne

In[383]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGemodels, DGemodelsDc}, 2]
```



Out[383]=



14.4 Dyskusja

Przy dyskretyzacji biegunów koniecznym zabiegiem okazało się nie dyskretyzowanie współczynnika stojącego przed całością filtra, gdyż przy niskich częstotliwościach i wysokich rzędach filtrów stawał się on bardzo mały, a po dyskretyzacji = 0. Przy tych założeniach większość filtrów okazała się stabilna przy dyskretyzacji za pomocą 6 bitów.

Okazuje się jednak, że dyskretyzacja na poziomie biegunów prowadzi do obniżenia rzędu filtra w przypadku filtrów eliptycznych (tj. zera i bieguny znajdujące się blisko siebie zaczynają wzajemnie się

skracać)

Dyskretyzacja współczynników ma bardzo podobny problem widoczny przy niskich częstotliwościach filtru Butterwortha i Chebysheva 1, z powodu znikającego licznika znika cały filtr.
Ponadto widać, iż filtry eliptyczne powyżej 3 rzędu robią się bardzo niestabilne (niektóre nie są zaznaczone na czerwono bo filtr znikł z powodów wspomnianych wyżej)

W świetle tych wyników oraz faktu, iż na urządzeniu będziemy dyskretyzować współczynniki, a nie bieguny, proponuję zastosować dyskretyzację zer i biegunów z osobna, tak by zapobiec zerowaniu się licznika.
Wykonane w celu sprawdzenia tej propozycji symulacje przedstawiam poniżej.

14.5 Dyskretyzacja na poziomie współczynników zer i biegunów z osobna

14.5.1 Definicje

Tutaj modyfikuję funkcję DiscretizeComplexList

In[422]:=
```
DiscretizeList[x_, max_, bits_] :=
  If[ # ≥ 0 , 1, -1] Round[ (Abs[#] / max) (Power[2, bits - 1] - 1)] max / (Power[2, bits - 1] - 1) & /@ x;
DiscretizeComplexList2[x_, y_, bits_] := Module[{xD, yD, coeffD, max},
   If[y == {}, {},
    max = Max[Abs[Join[Re@x, Im@x]]];
    xD = DiscretizeList[Re@x, max, bits] + i DiscretizeList[Im@x, max, bits];
    max = Max[Abs[Join[Re@y, Im@y]]];
    yD = DiscretizeList[Re@y, max, bits] + i DiscretizeList[Im@y, max, bits];
    {xD, yD}]
  ];
DiscretizeModelCoeffs2[tf_, bits_] := Module[{zeros, poles, zerosD, polesD},
      zeros = CoefficientList[Numerator[TransferFunctionExpand[tf][z]][[1, 1]], z];
   poles = CoefficientList[Denominator[TransferFunctionExpand[tf][z]][[1, 1]], z];
   {zerosD, polesD} = DiscretizeComplexList2[zeros, poles, bits];
      Chop[FromDigits[Reverse[zerosD], z] / (FromDigits[Reverse[polesD], z])]
  ];
```

14.5.2 Testy

In[363]:=
```
{a, c} = DiscretizeComplexList[{1, 2, 3}, {i, i 2, i 3}, 2]
```

Out[363]=
```
{{0, 3, 3}, {0, 3 i, 3 i}}
```

Dyskretyzujemy na poziomie filtrów cyfrowych

In[371]:=
```
buttModel = TransferFunctionExpand@DGbmodels[[2, 2]]
```

Out[371]=
$$\left(\frac{0.0302379 + 0.0604758\,\dot{z} + 0.0302379\,\dot{z}^2}{(0.572371 + 0.\,i) - (1.45142 + 0.\,i)\,\dot{z} + \dot{z}^2}\right)_{\mathcal{T}}$$

In[373]:=
```
polesButt = TransferFunctionPoles[buttModel][[1, 1]]
```

Out[373]=
$$\{0.72571 - 0.213814\,i,\ 0.72571 + 0.213814\,i\}$$

In[374]:=
```
buttModelD = N@DiscretizeModelCoeffs[buttModel, bity]
```

Out[374]=
$$\frac{0.04682 + 0.04682\,z + 0.04682\,z^2}{0.56184 - 1.45142\,z + 0.98322\,z^2}$$

In[375]:=
```
polesButtD = ExtractPoles[buttModelD]
```

Out[375]=
$$\{0.738095 - 0.16323\,i,\ 0.738095 + 0.16323\,i\}$$

14.5.3 Dyskretyzacja

In[425]:=
```
DGbmodelsDc2 = Map[DiscretizeModelCoeffs2[#, bity] &, DGbmodels, {2}];
```

In[426]:=
```
DGc1modelsDc2 = Map[DiscretizeModelCoeffs2[#, bity] &, DGc1models, {2}];
```

In[427]:=
```
DGc2modelsDc2 = Map[DiscretizeModelCoeffs2[#, bity] &, DGc2models, {2}];
```

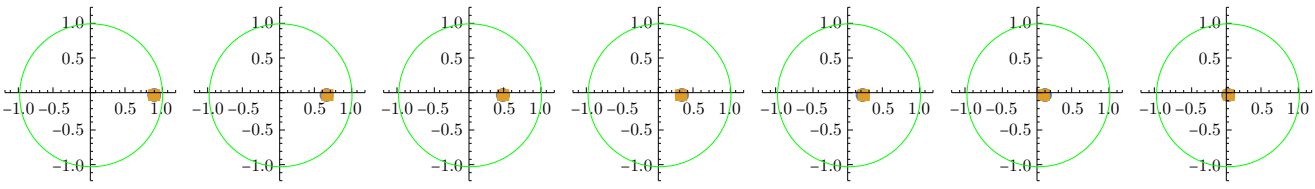
In[428]:=
```
DGemodelsDc2 = Map[DiscretizeModelCoeffs2[#, bity] &, DGemodels, {2}];
```

14.5.4 Porównanie położenia biegunów

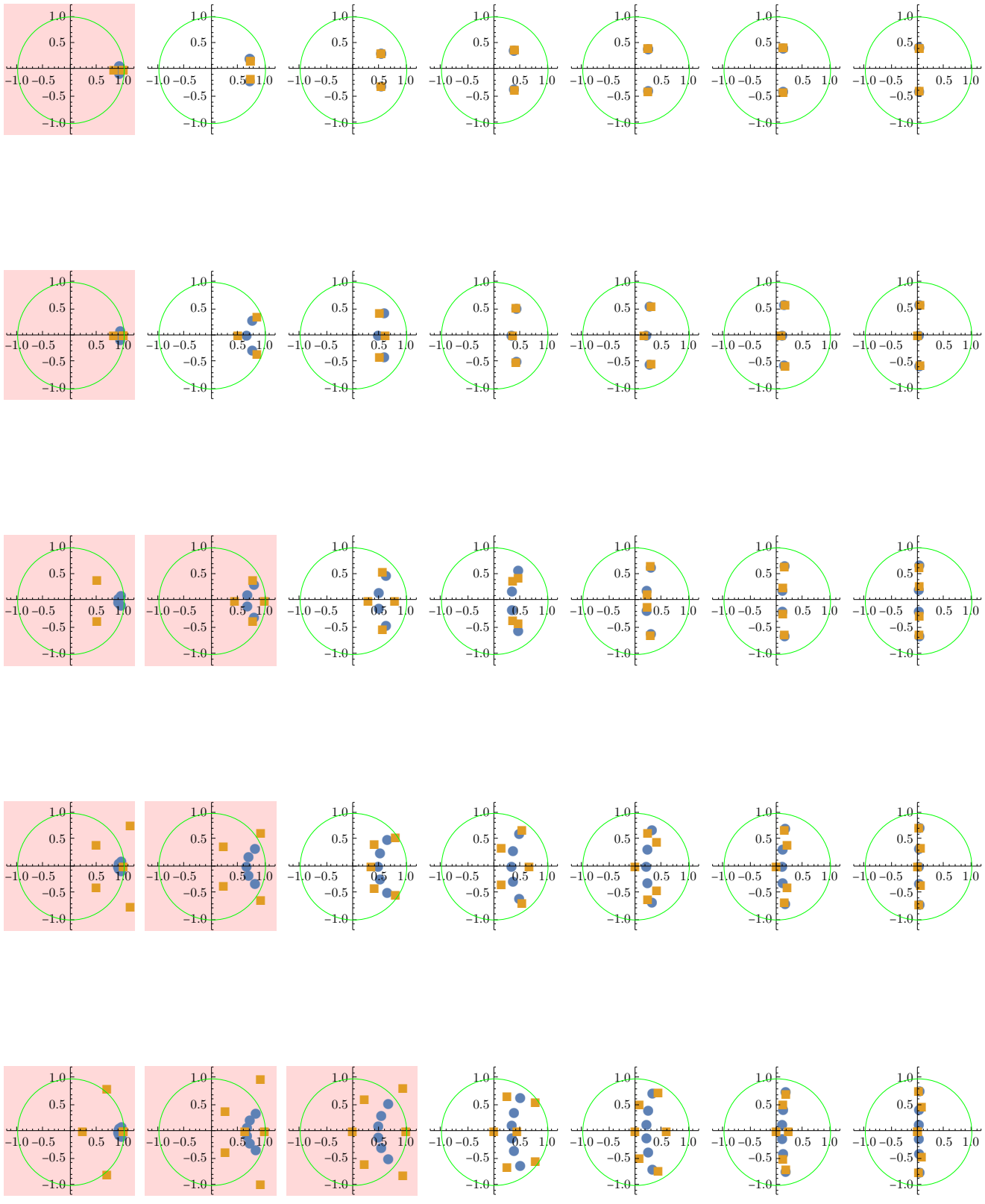
★ Butterworth

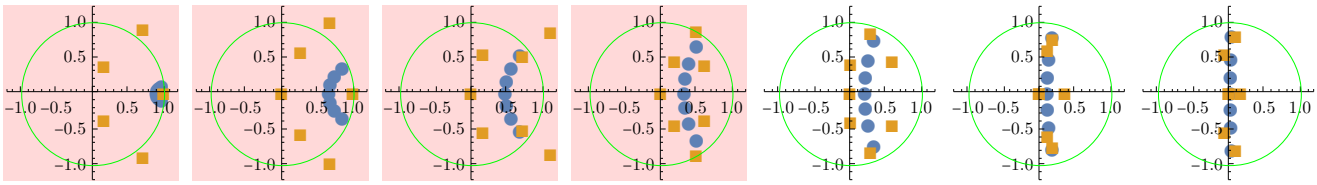
W prawo rośnie częstotliwość, w dół rośnie rząd filtra.

In[429]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGbmodels, DGbmodelsDc2}, 2]
```



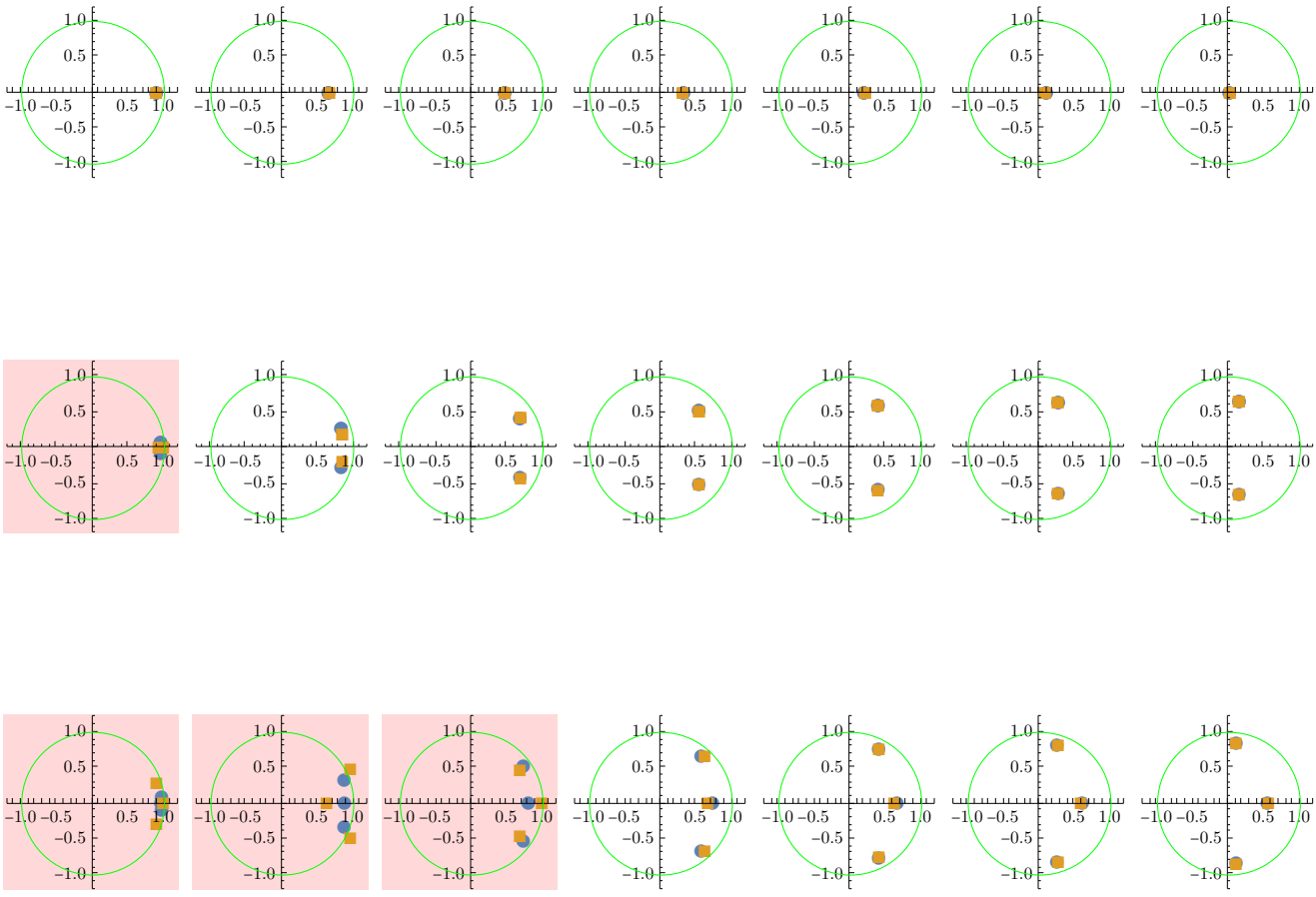
Out[429]=





★ Chebyshev 1

In[430]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGc1models, DGc1modelsDc2}, 2]
```

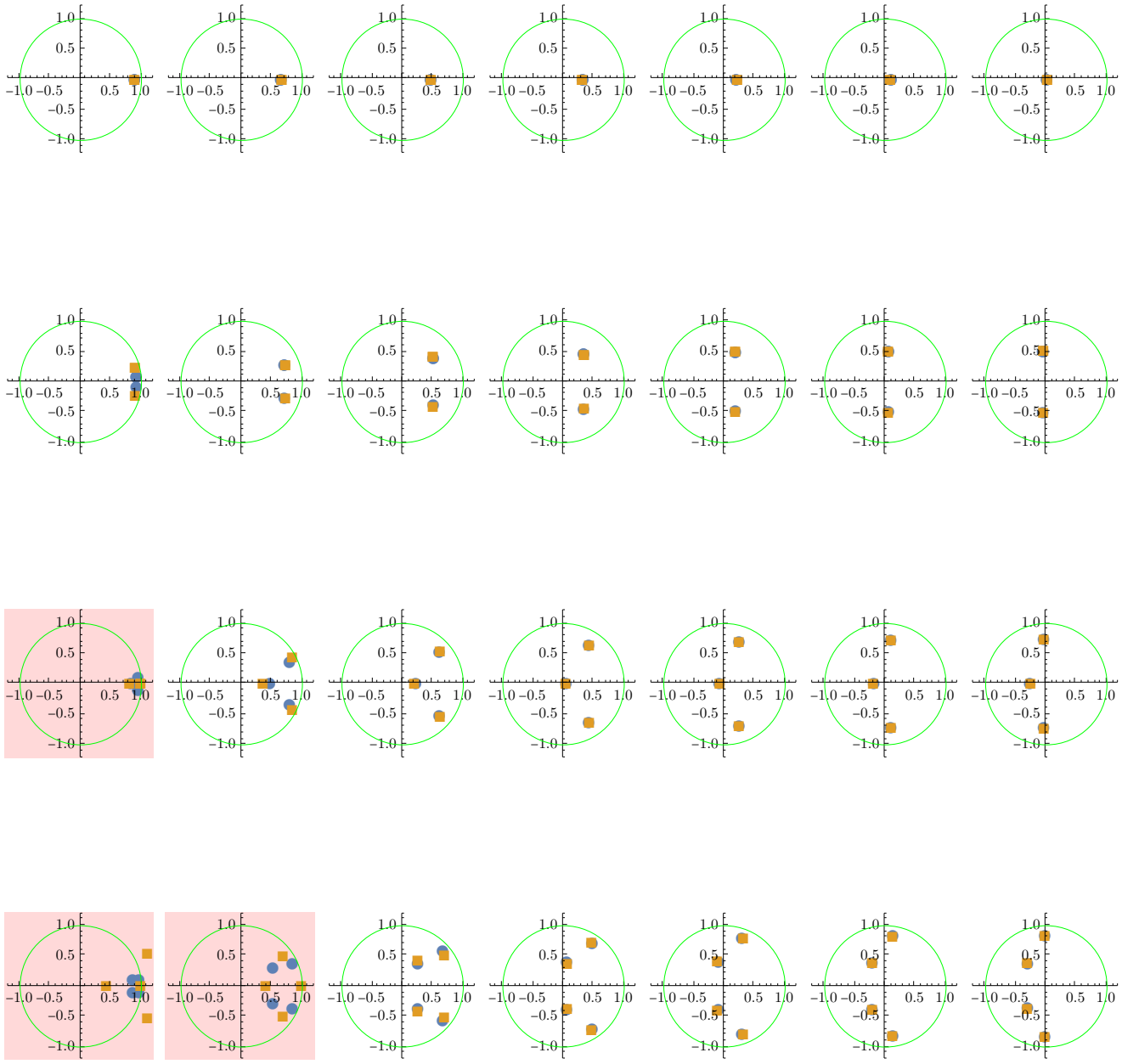


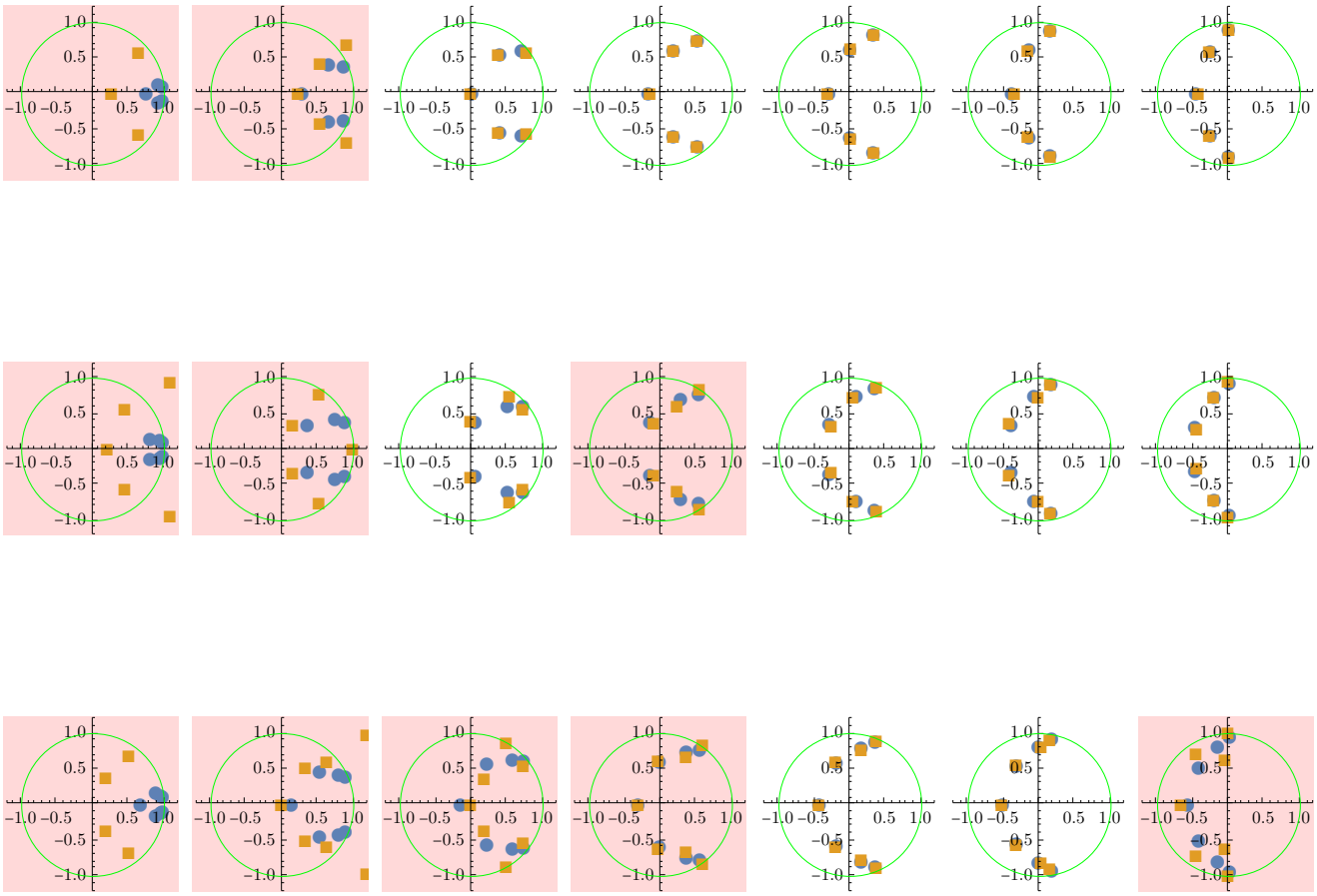
Out[430]=

★ Chebyshev 2

In[431]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGc2models, DGc2modelsDc2}, 2]
```

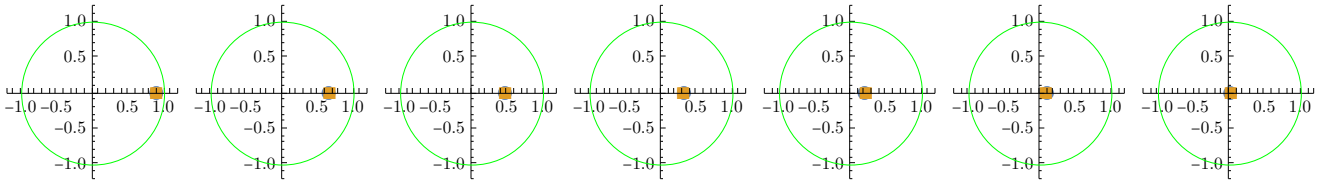
Out[431]=



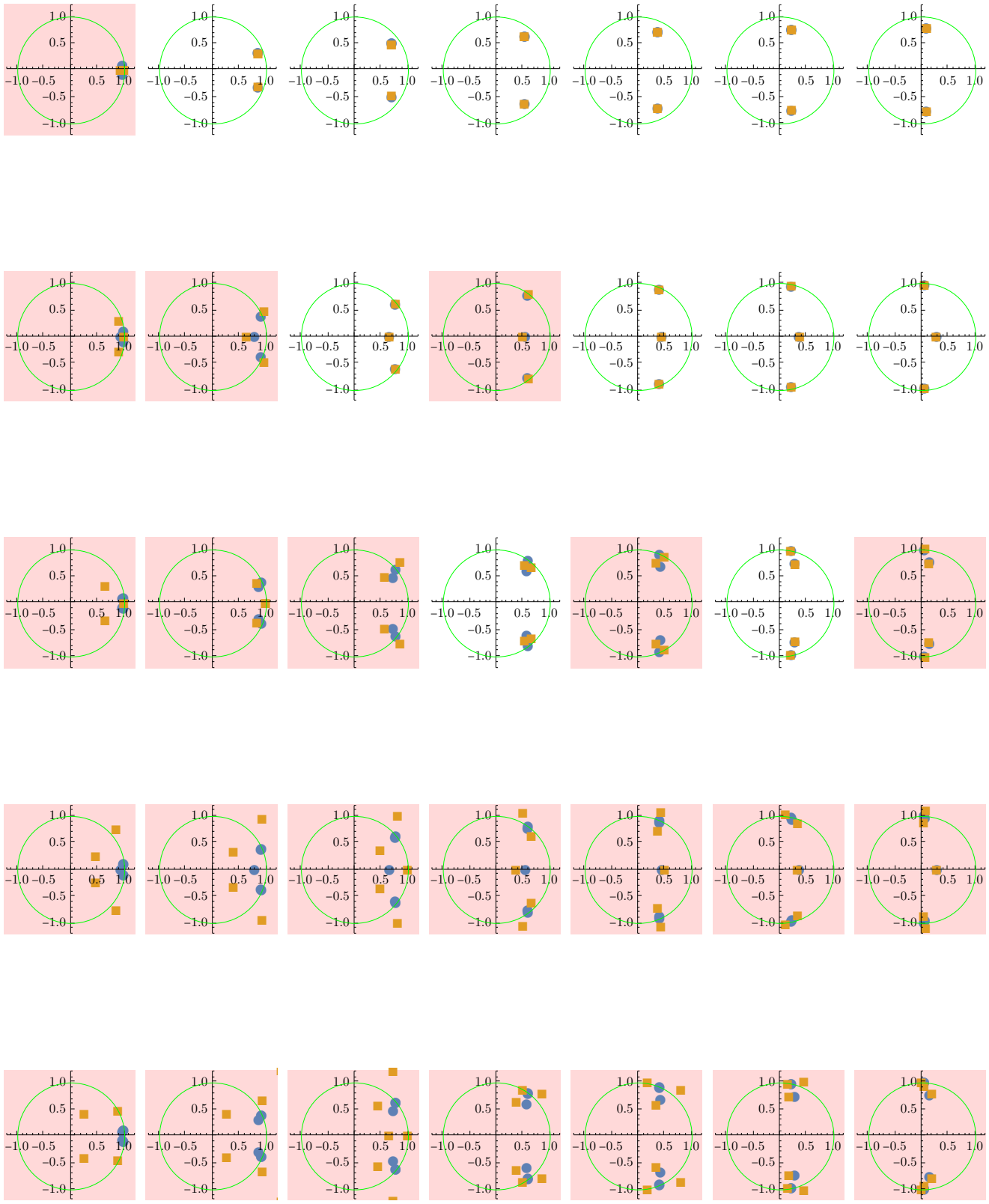


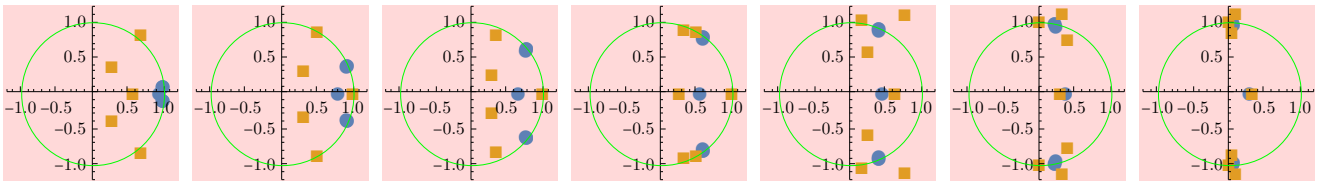
★ Eliptyczne

In[432]:=
```
Grid@MapThread[PlotPoles[TransferFunctionPoles[#1][[1, 1]], ExtractPoles[#2]] &,
  {DGemodels, DGemodelsDc2}, 2]
```



Out[432]=





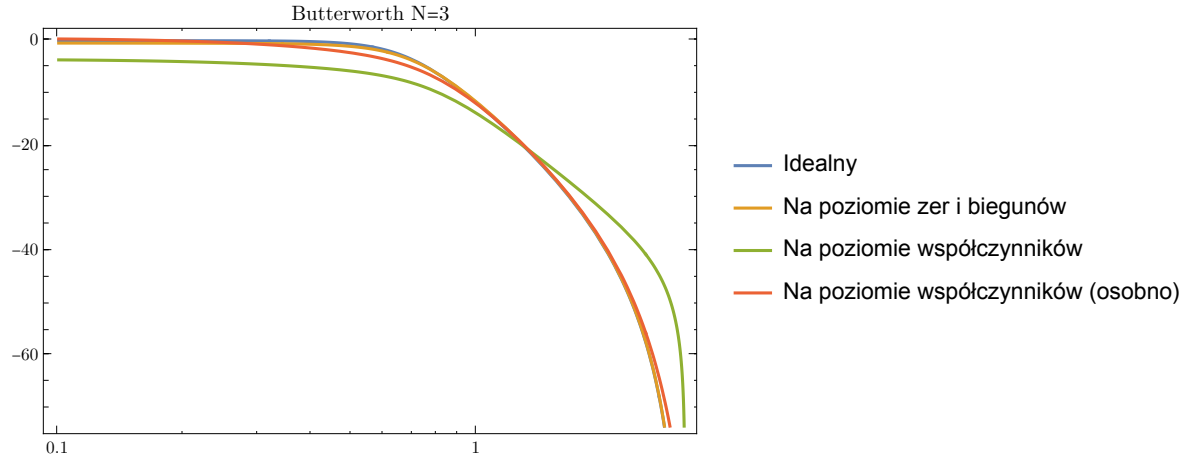
15 Podsumowanie

Wbrew pozorom podzielenie procesu dyskretyzacji na współczynniki licznika i mianownika nie przyniosło wielkich zmian - filtry który znikały, stały się po prostu niestabilne.
Używając Bode Plot (ilustrujący kwadrat amplitudy (dB) w zależności od ω) można zademonstrować, że lepsze odwzorowanie filtru zależy właśnie od przyjętej strategii dyskretyzacji współczynników.

Poniżej znajdują się przykładowe Bode Ploty dla filtrów Butterwortha trzeciego rzędu i częstości równej 1.

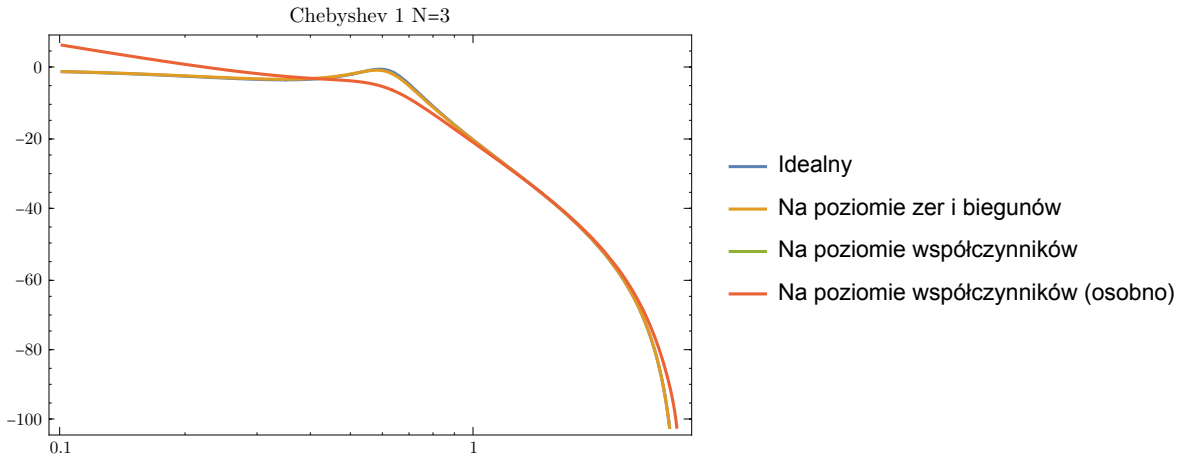
In[466]:=
```
BodePlot[{DGbmodels[[3, 3]][e^(I ω)], DGbmodelsD[[3, 3]] /. {z -> e^(I ω)},
  DGbmodelsDc[[3, 3]] /. {z -> e^(I ω)}, DGbmodelsDc2[[3, 3]] /. {z -> e^(I ω)}}, {ω, 0.1, π},
 PlotLayout → "Magnitude",
 PlotLegends → {"Idealny", "Na poziomie zer i biegunów", "Na poziomie współczynników",
   "Na poziomie współczynników (osobno)"}, PlotLabel → "Butterworth N=3"]
```

Out[466]=



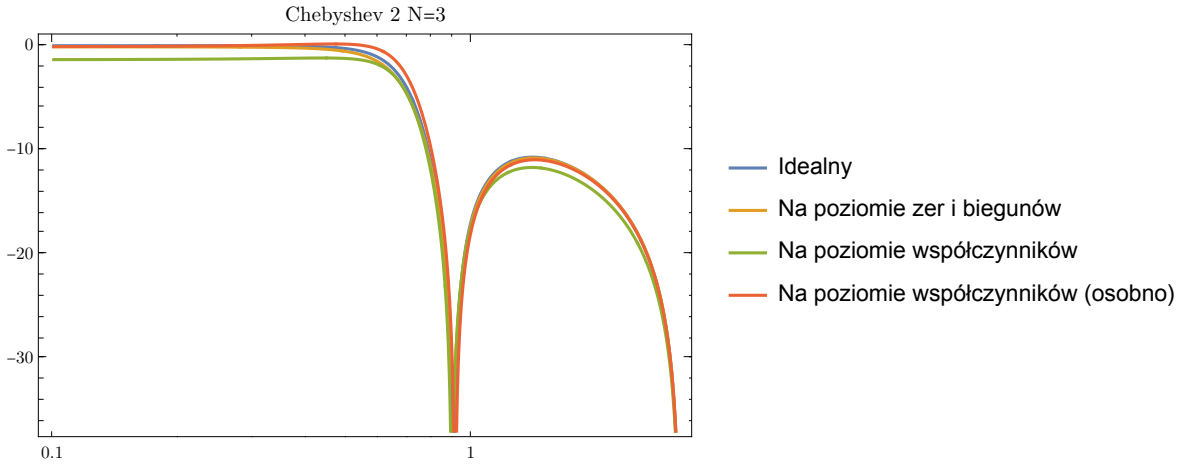
In[500]:=
```
BodePlot[{DGc1models[[3, 3]][e^(i ω)], DGc1modelsD[[3, 3]] /. {z -> e^(i ω)},
  DGc1modelsDc[[3, 3]] /. {z -> e^(i ω)}, DGc1modelsDc2[[3, 3]] /. {z -> e^(i ω)}}, {ω, 0.1, π},
 PlotLayout → "Magnitude",
 PlotLegends → {"Idealny", "Na poziomie zer i biegunów", "Na poziomie współczynników",
   "Na poziomie współczynników (osobno)"}, PlotLabel → "Chebyshev 1 N=3"]
```

Out[500]=



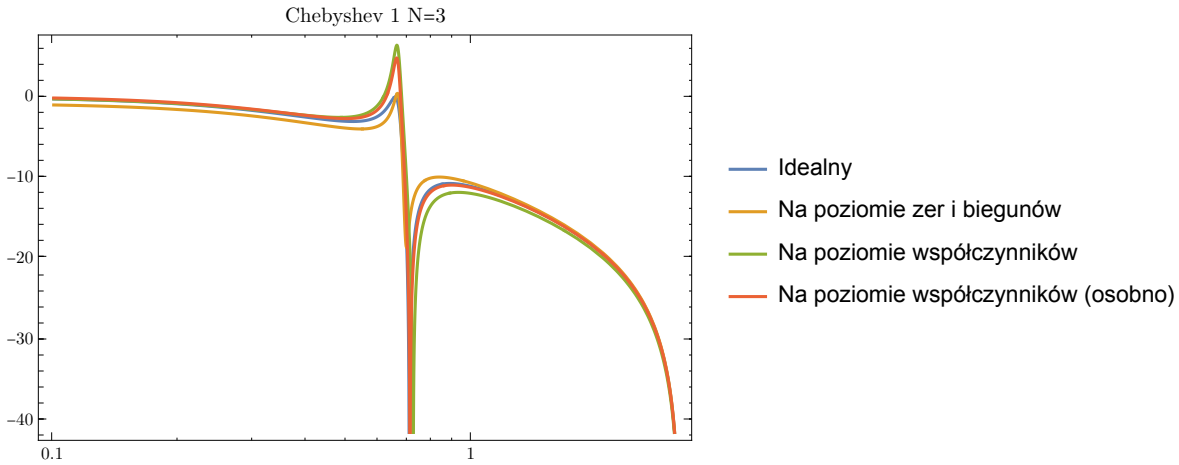
In[501]:=
```
BodePlot[{DGc2models[[3, 3]][e^(i ω)], DGc2modelsD[[3, 3]] /. {z -> e^(i ω)},
  DGc2modelsDc[[3, 3]] /. {z -> e^(i ω)}, DGc2modelsDc2[[3, 3]] /. {z -> e^(i ω)}}, {ω, 0.1, π},
 PlotLayout → "Magnitude",
 PlotLegends → {"Idealny", "Na poziomie zer i biegunów", "Na poziomie współczynników",
   "Na poziomie współczynników (osobno)"}, PlotLabel → "Chebyshev 2 N=3"]
```

Out[501]=



In[502]:=
```
BodePlot[{DGemodels[[3, 3]][e^(i ω)], DGemodelsD[[3, 3]] /. {z -> e^(i ω)},
  DGemodelsDc[[3, 3]] /. {z -> e^(i ω)}, DGemodelsDc2[[3, 3]] /. {z -> e^(i ω)}}, {ω, 0.1, π},
 PlotLayout → "Magnitude",
 PlotLegends → {"Idealny", "Na poziomie zer i biegunów", "Na poziomie współczynników",
   "Na poziomie współczynników (osobno)"}, PlotLabel → "Chebyshev 1 N=3"]
```

Out[502]=



Jeśli trzeba można wyrysować wszystkie tym poleceniem.

```
Grid@
 MapThread[BodePlot[{#1[e^(i ω)], #2 /. {z -> e^(i ω)}, #3 /. {z -> e^(i ω)}, #4 /. {z -> e^(i ω)}},
    {ω, 0.01, π}, PlotLayout → "Magnitude"] &, {DGbmodels, DGbmodelsD, DGbmodelsDc, DGbmodelsDc2},
  2]
```

Out[440]= $Aborted

Wybrana przeze mnie wcześniej procedura dyskretyzacji na poziomie zer i biegunów, pomimo swoich pozornych zalet obarczona jest jednak dodatkowym błędem, wynikającym z wymnożenia zdyskretyzowanych zer/biegunów którego należałoby by dokonać nie na liczbach rzeczywistych, lecz na liczbach n-bitowych.
Prawdopodobnie to dyskwalifikuje tę metodę.