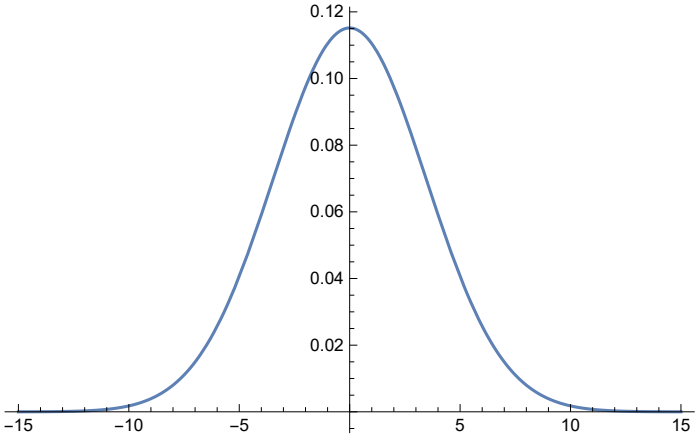
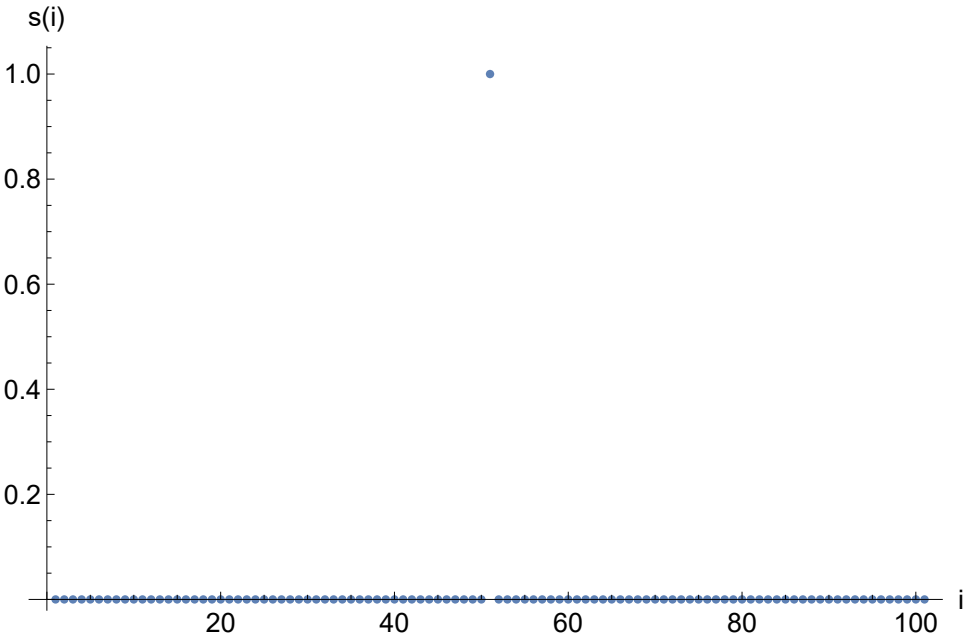
FED is motivated by symmetric filtering in the 1D case. The goal is to use special filters to approximate a Gaussian function. Applying a Gaussian to a signal is the same like applying a diffusion to the signal. The special filters are symmetric filters with $\sum w_i = 1$. Multiple iterations of these filters approximate a Gaussian function. It can be shown that applying a FED cycle is the same like applying the symmetric filter. This also means that multiple FED cycles approximate a Gaussian and this in term means that FED cycles are an approximation of the diffusion equation.

```
σ = √12 ;
ϕ[x_] := PDF[NormalDistribution[0, σ], x];
Plot[ϕ[x], {x, -15, 15}, PlotRange → Full]
```



Create a signal with one peak in the middle.

```
signalLength = 101;
f = ConstantArray[0, signalLength];
fMid = ⌈signalLength / 2⌉;
f[[fMid]] = 1;
ListPlot[f,
 AxesLabel → {"i", "s(i)"},
 ImageSize → 500,
 BaseStyle → {FontSize → 14}
]
```



Create a box filter

```
wBox[n_] := 1/(2 n + 1);
nBox = 3;
kernelBox = ConstantArray[wBox[nBox], 2 * nBox + 1]
kernelBox // Length
```

$$\left\{\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}\right\}$$

7

```
wBin[n_, k_] := 1/4^n Binomial[2 n, n + k];
nBin = 8;
kernelBin = Table[wBin[nBin, k], {k, -nBin, nBin}]
kernelBin // Length
```

$$\left\{\frac{1}{65\,536}, \frac{1}{4096}, \frac{15}{8192}, \frac{35}{4096}, \frac{455}{16\,384}, \frac{273}{4096}, \frac{1001}{8192}, \frac{715}{4096}, \frac{6435}{32\,768}, \frac{715}{4096}, \frac{1001}{8192}, \frac{273}{4096}, \frac{455}{16\,384}, \frac{35}{4096}, \frac{15}{8192}, \frac{1}{4096}, \frac{1}{65\,536}\right\}$$

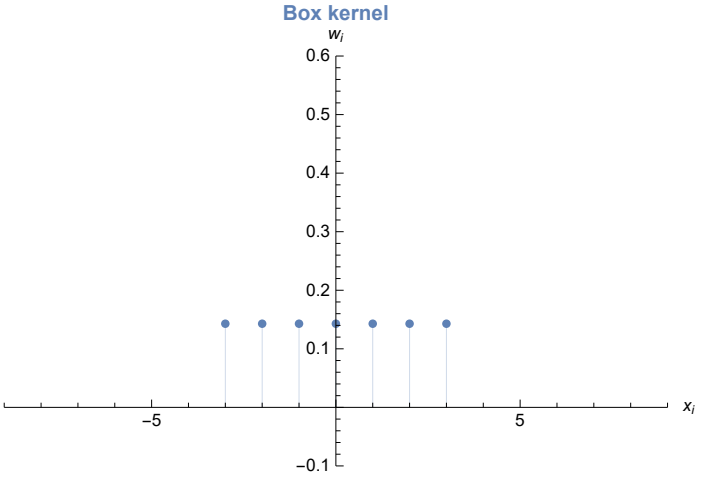
17

```
wMV[n_, k_] := 1/2 * {0  Abs[k] ≠ n
                      1  Abs[k] == n;
nMV = 2;
kernelMV = Table[wMV[nMV, k], {k, -nMV, nMV}]
kernelMV // Length
```

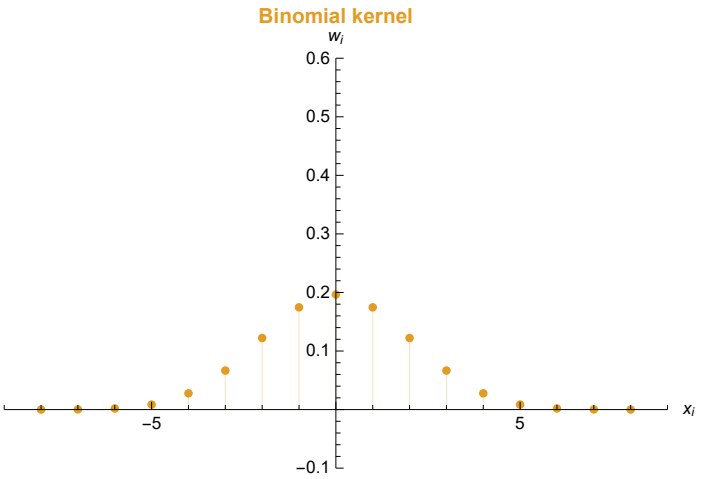
$$\left\{\frac{1}{2}, 0, 0, 0, \frac{1}{2}\right\}$$

5

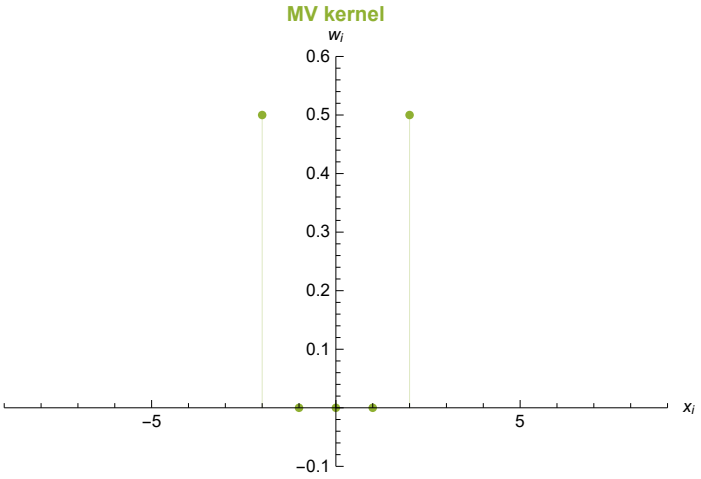
```
ListPlot[kernelBox, DataRange → {-nBox, nBox}, PlotRange → {{-nBin - 1, nBin + 1}, {-0.1, 0.6}},
 PlotLabel → Style["Box kernel", FontWeight → Bold, FontColor → ■],
 Filling → Axis, AxesLabel → {"xi", "wi"}]
```




```
ListPlot[kernelBin, DataRange → {-nBin, nBin}, PlotRange → {{-nBin - 1, nBin + 1}, {-0.1, 0.6}},
 PlotStyle → ■, PlotLabel → Style["Binomial kernel", FontWeight → Bold, FontColor → ■],
 Filling → Axis, AxesLabel → {"xi", "wi"}]
```




```
ListPlot[kernelMV, DataRange → {-nMV, nMV}, PlotRange → {{-nBin - 1, nBin + 1}, {-0.1, 0.6}},
 PlotStyle → , PlotLabel → Style["MV kernel", FontWeight → Bold, FontColor → ],
 Filling → Axis, AxesLabel → {"xi", "wi"}]
```



$$\text{ϴBox}[n\_] := \frac{n^2 + n}{6};$$

$$\text{ϴBin}[n\_] := \frac{n}{4};$$

$$\text{ϴMV}[n\_] := \frac{n^2}{2};$$

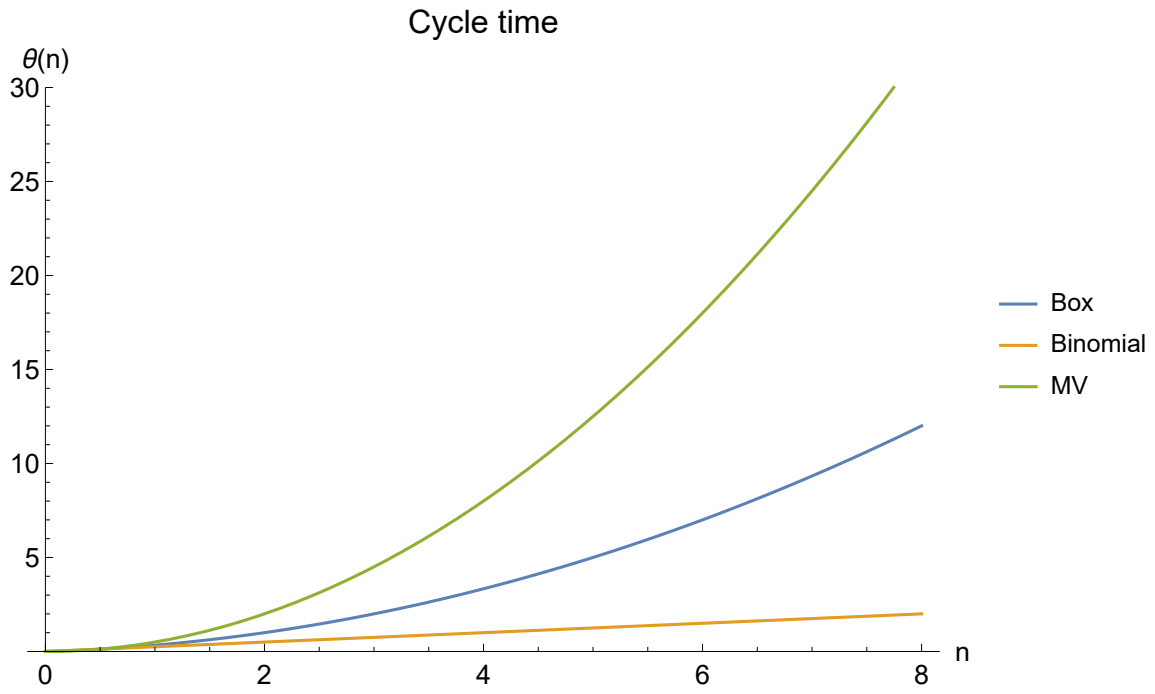
```
Solve[ϴBox[n] == 2, n] // N
Solve[ϴBin[n] == 2, n]
Solve[ϴMV[n] == 2, n] // N
```

```
{{n → -4.}, {n → 3.}}
```

```
{{n → 8}}
```

```
{{n → -2.}, {n → 2.}}
```

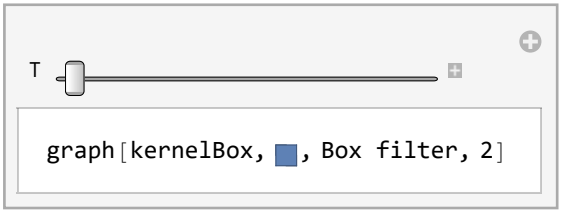
```
Plot[{ϴBox[n], ϴBin[n], ϴMV[n]}, {n, 0, nBin},
 PlotRange → {0, 30},
 AxesLabel → {"n", "ϴ(n)"},
 PlotLegends → {"Box", "Binomial", "MV"},
 PlotLabel → "Cycle time",
 ImageSize → 500,
 BaseStyle → {FontSize → 14}
]
```


Cycle time
θ(n)
30
25
20
15
10
5
0
2
4
6
8
n
Box
Binomial
MV

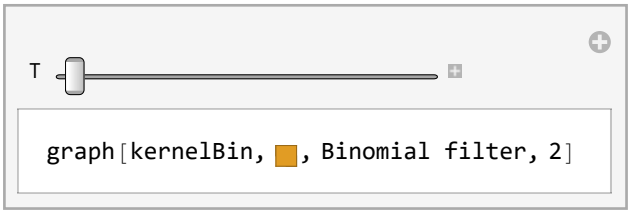
```
applyKernel[kernel_, iter_] := Module[{res},
  res = f;
  Do[
   res = ListConvolve[kernel, ArrayPad[res, ⌊Length[kernel]/2⌋, "Reversed"]] // N;
   , {i, 1, iter}
  ];

  res
 ]

error[signal_, base_] := Module[{n},
  n = Length[signal];

  ∑_{i=1}^{101} Abs[signal[i] - base[i]]
 ]

graph[kernel_, color_, label_, iter_] := Module[{convolvedSignal, dist, errorSum},
  convolvedSignal = applyKernel[kernel, iter * 0.5];
  (* The resulting signal can be interpreted as a 1D discrete probabilistc function *)
  dist = EmpiricalDistribution[convolvedSignal → Range[1, Length[convolvedSignal]]];
  errorSum = error[PDF[dist], PDF[NormalDistribution[Mean[dist], StandardDeviation[dist]]]];

  Show[
   Plot[PDF[NormalDistribution[Mean[dist], StandardDeviation[dist]], x], {x, 0, 100},
    PlotStyle → ■,
    PlotRange → All,
    AxesLabel → {"x_i", "w_i"},
    PlotLegends → {"Gaussian"},
    PlotLabel → label
      <> " (μ = "
      <> ToString[Round[Mean[dist]], StandardForm]
      <> ", σ^2 = " <> ToString[Round[Variance[dist]], StandardForm]
      <> ", T = " <> ToString[iter * 2]
      <> ", E = " <> ToString[errorSum]
      <> ")",
    ImageSize → 500,
    BaseStyle → {FontSize → 14}
   ],
   DiscretePlot[PDF[dist, x], {x, 0, 100},
    PlotRange → Full,
    PlotLegends → {"Approximation"},
    PlotStyle → color,
    PlotMarkers → {Automatic, 7}
   ]
  ]
 ]
```

Multiple iterations (3 in this case) of the three used filters indeed approximate a Gaussian. The size of the kernels is chosen so that the resulting variance is always the same.

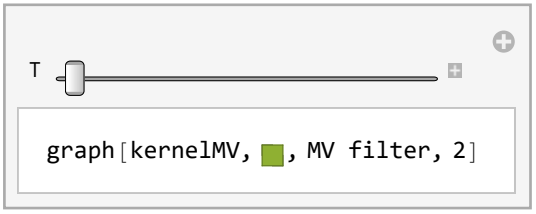
```
Manipulate[
 graph[kernelBox, ■, "Box filter", T]
 , {T, 2, 6, 2}]
```




```
Manipulate[
 graph[kernelBin, , "Binomial filter", T]
 , {T, 2, 6, 2}]
```




```
Manipulate[
 graph[kernelMV, , "MV filter", T]
 , {T, 2, 6, 2}]
```




```
(*Export[FileNameJoin[{NotebookDirectory[],"frames/filter=0T=0.png"}],
 Table[graph[kernelBox,,"Box filter",T],{T,2,6,2}],"VideoFrames",Antialiasing→True];
Export[FileNameJoin[{NotebookDirectory[],"frames/filter=1T=0.png"}],
 Table[graph[kernelBin,,"Binomial filter",T],{T,2,6,2}],"VideoFrames",Antialiasing→True];
Export[FileNameJoin[{NotebookDirectory[],"frames/filter=2T=0.png"}],
 Table[graph[kernelMV,,"MV filter",T],{T,2,6,2}],"VideoFrames",Antialiasing→True];*)
```

$$A = \text{Table}\left[\begin{cases} -1 & i == 1 \,\&\&\, j == 1 \,||\, i == \text{signalLength} \,\&\&\, j == \text{signalLength} \\ -2 & i == j \\ 1 & i == j+1 \,||\, j == i+1 \\ 0 & \text{True} \end{cases}, \{i, 1, \text{signalLength}\}, \{j, 1, \text{signalLength}\}\right];$$

$$\tau\text{Max} = \frac{1}{2};$$

$$\tau\text{Box}[n\_, i\_] := \tau\text{Max} * \frac{1}{2 * \text{Cos}\left[\pi * \frac{2*i+1}{4*n+2}\right]^2};$$

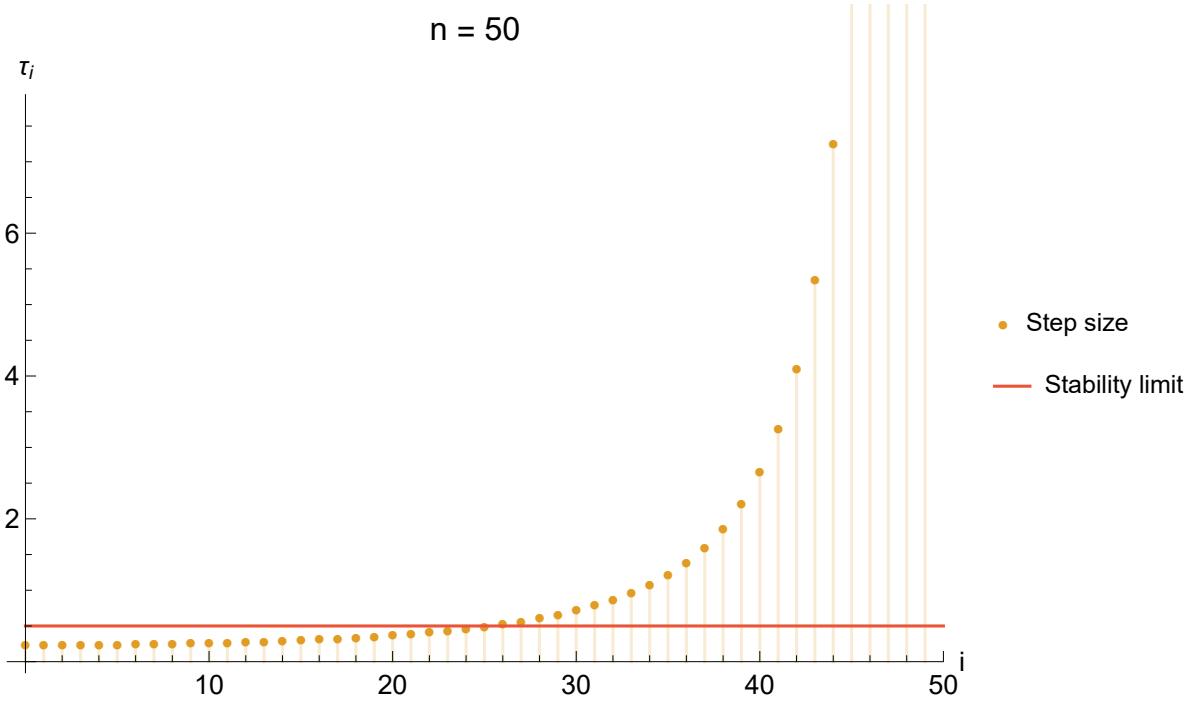
$$\tau\text{Bin}[n\_, i\_] = \tau\text{Max} * \frac{1}{2};$$

$$\tau\text{MV}[n\_, i\_] := \tau\text{Max} * \frac{1}{2 * \text{Cos}\left[\pi * \frac{2i+1}{4n}\right]^2}$$

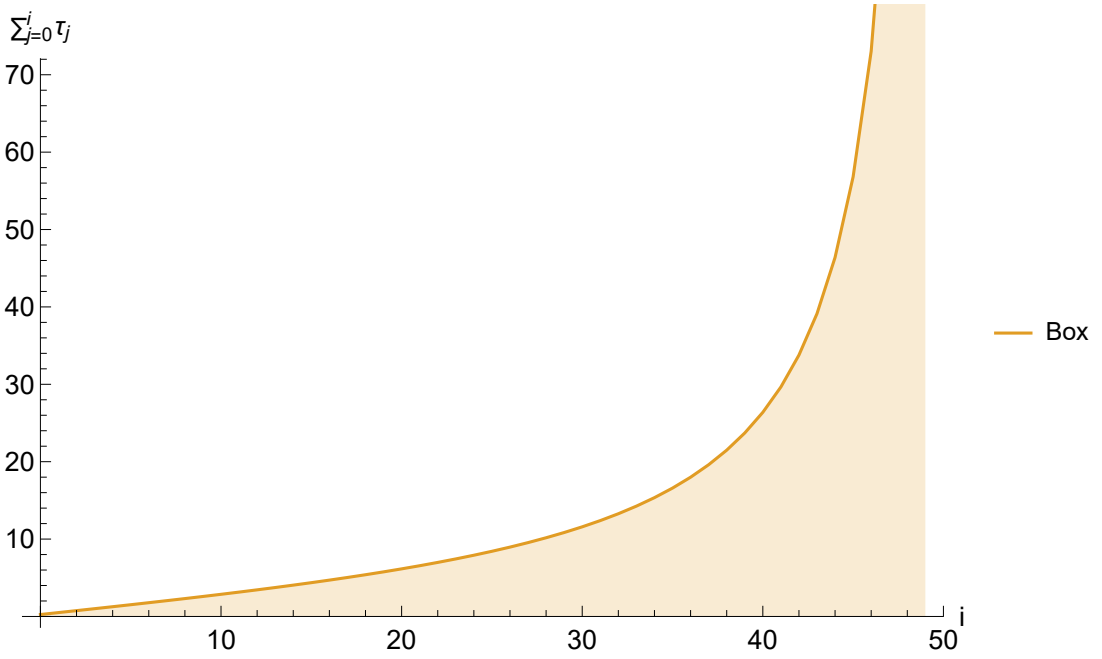
Some of the step sizes exceed the stability limit.

```
nHigher = 50;
```

```
Show[
 DiscretePlot[{τBox[nHigher, i]}, {i, 0, nHigher - 1},
  AxesLabel → {"i", "τᵢ"},
  PlotLabel → "n = " <> ToString[nHigher],
  PlotLegends → {"Step size"},
  PlotMarkers → {Automatic, 7},
  PlotStyle → ■,
  ImageSize → 500,
  BaseStyle → {FontSize → 14}
 ],
 Plot[0.5, {x, 0, nHigher},
  PlotStyle → ■,
  PlotLegends → {"Stability limit"}
 ]
]
```



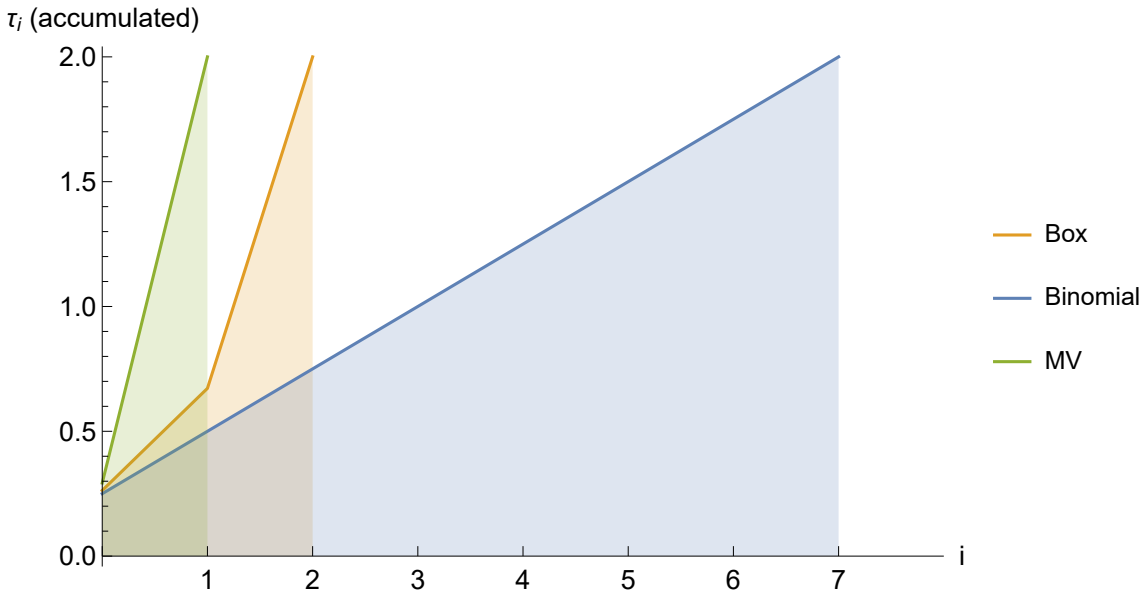

```
DiscretePlot[{∑_{j=0}^{i} τBox[nHigher, j]}, {i, 0, nHigher - 1},
 Joined → True,
 PlotStyle → ■,
 AxesLabel → {"i", "∑_{j=0}^{i} τⱼ"},
 PlotLegends → {"Box"},
 ImageSize → 500,
 BaseStyle → {FontSize → 14}
]
```



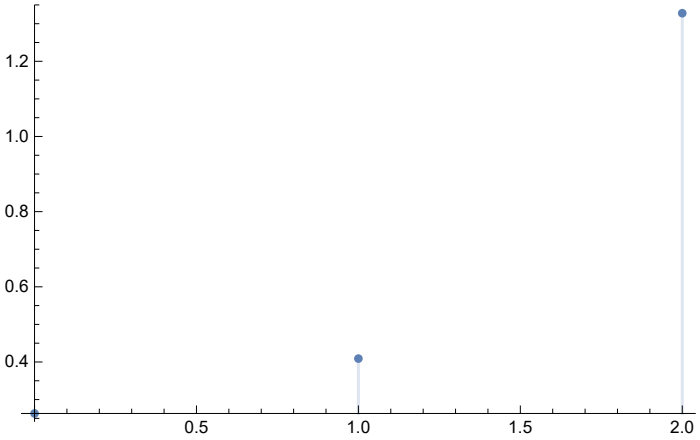

```
Sum[τBox[nHigher, j] // N, {j, 0, nHigher - 1}]
```

425.

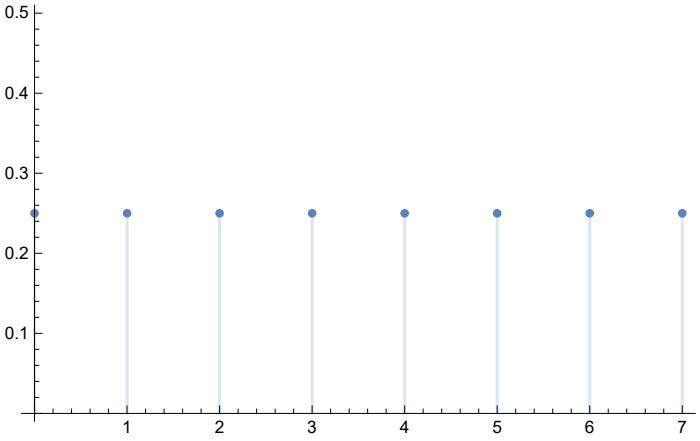
```
Show[
 DiscretePlot[{Sum[τBox[nBox, j], {j, 0, i}]}, {i, 0, nBox - 1},
  Joined → True,
  Ticks → {Range[0, nBin - 1], Automatic},
  PlotStyle → ■,
  PlotRange → {{0, nBin}, Automatic},
  AxesLabel → {"i", "τ_i (accumulated)"},
  PlotLegends → {"Box"},
  ImageSize → 500,
  BaseStyle → {FontSize → 14}
 ],
 DiscretePlot[{Sum[τBin[nBin, j], {j, 0, i}]}, {i, 0, nBin - 1}, Joined → True, PlotLegends → {"Binomial"}],
 DiscretePlot[{Sum[τMV[nMV, j], {j, 0, i}]}, {i, 0, nMV - 1}, Joined → True, PlotStyle → ■, PlotLegends → {"MV"}]
]
```



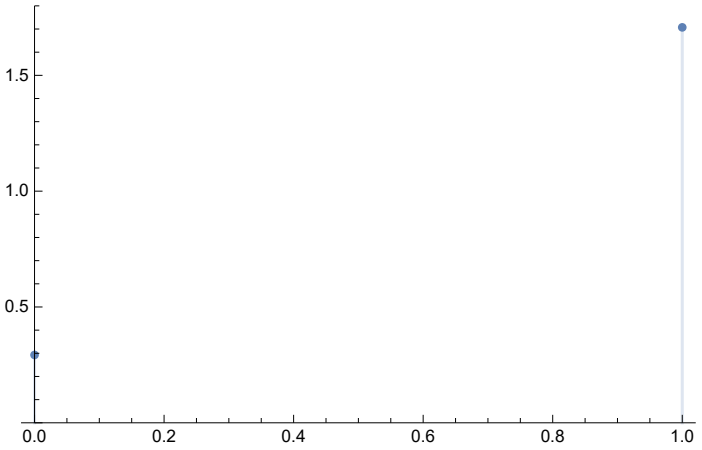

```
DiscretePlot[{τBox[nBox, i]}, {i, 0, nBox - 1}]
```




```
DiscretePlot[{τBin[nBin, i]}, {i, 0, nBin - 1}]
```




```
DiscretePlot[{τMV[nMV, i]}, {i, 0, nMV - 1}, PlotRange → {0, 1.8}]
```




```
calcFED[signal_, n_, τ_] := Module[{imRes},
  imRes = signal;
  Do[
   imRes = (IdentityMatrix[signalLength] + τ[n, i] * A).imRes
   , {i, 0, n - 1}];

  imRes
 ]

applyFED[n_, τ_] := Module[{res},
  res = f;
  Do[
   res = calcFED[res, n, τ];
   , {i, 1, 3}
  ];

  res
 ]
```

Applying FED cycles is indeed equivalent of applying kernels.

```
applyFED[nBox, τBox] // N
applyKernel[kernelBox] // N
```

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.00291545, 0.00874636,
 0.0174927, 0.0291545, 0.0437318, 0.0612245, 0.0816327, 0.0962099, 0.104956, 0.107872,
 0.104956, 0.0962099, 0.0816327, 0.0612245, 0.0437318, 0.0291545, 0.0174927, 0.00874636,
 0.00291545, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
```

```
applyKernel[{0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857}]
```

```
applyFED[nBin, τBin] // N
applyKernel[kernelBin] // N
```

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 3.55271 × 10^-15, 1.7053 × 10^-13, 4.00746 × 10^-12, 6.14477 × 10^-11, 6.91287 × 10^-10, 6.08333 × 10^-9,
 4.35972 × 10^-8, 2.61583 × 10^-7, 1.34061 × 10^-6, 5.95828 × 10^-6, 0.0000232373, 0.0000802743,
 0.000247512, 0.000685419, 0.00171355, 0.00388404, 0.00801083, 0.0150792, 0.0259698, 0.0410049,
 0.0594571, 0.0792761, 0.0972934, 0.109984, 0.114567, 0.109984, 0.0972934, 0.0792761, 0.0594571,
 0.0410049, 0.0259698, 0.0150792, 0.00801083, 0.00388404, 0.00171355, 0.000685419, 0.000247512,
 0.0000802743, 0.0000232373, 5.95828 × 10^-6, 1.34061 × 10^-6, 2.61583 × 10^-7, 4.35972 × 10^-8,
 6.08333 × 10^-9, 6.91287 × 10^-10, 6.14477 × 10^-11, 4.00746 × 10^-12, 1.7053 × 10^-13, 3.55271 × 10^-15, 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
```

```
applyKernel[{0.0000152588, 0.000244141, 0.00183105, 0.00854492,
  0.027771, 0.0666504, 0.122192, 0.174561, 0.196381, 0.174561, 0.122192,
  0.0666504, 0.027771, 0.00854492, 0.00183105, 0.000244141, 0.0000152588}]
```

```
applyFED[nMV, τMV] // N // Chop
applyKernel[kernelMV] // N // Chop
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0.125, 0, 0, 0, 0.375, 0, 0, 0, 0.375, 0, 0, 0, 0.125, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

```
applyKernel[{0.5, 0, 0, 0, 0.5}]
```