Finite Difference Formulas Generated by an Interpolating Polynomial

Initialization Code (optional)

Manipulate

```
Manipulate[
 Module[{points, der, formula, plot, exact, header2, tbl, lte},

  points = makePoints[nPoints, finiteDifferenceType];

  {lte, der, formula, tbl, exact, plot} = mk[derivativeOrder, at, points, fun, f, h, x, plotType,
    nForApprox, n2ForApprox, n3ForApprox, xForApprox, yForApprox, finiteDifferenceType];

  header2 = Text@Grid[{
      {Item[TraditionalForm@Style[HoldForm[+##] &[formula, lte], Large], ItemSize → {49.5, 4}]}
    }, Frame → All, FrameStyle -> Directive[Thickness[.005], Gray],
    AllowScriptLevelChange → True, Spacings → {0, .2}];

  Text@Grid[{
     {header2, SpanFromLeft},
     {tbl, Grid[{{exact}, {plot}},
       Spacings → {0, 0.1}, Frame → All, FrameStyle -> Directive[Thickness[.005], Gray]]}
    }, Spacings → {0, 0}]
 ],

 Text@Grid[{{
    Grid[{{
      (******************)
      Grid[{{
        {RadioButtonBar[Dynamic[derivativeOrder, {derivativeOrder = #;

              nPoints = Which[
                finiteDifferenceType == "centered",
                If[derivativeOrder == 1 || derivativeOrder == 2, orderCentered + 1, orderCentered + 3],

                finiteDifferenceType == "forward", derivativeOrder + orderForward,

                finiteDifferenceType == "backward", derivativeOrder + orderBackward
               ]} &], {
            1 → "",
            2 → "",
            3 → "",
            4 → ""
           },
           Appearance → "Horizontal"], SpanFromLeft
        },
        {Style["f", Italic]"(1)",
         Style["f", Italic]"(2)",
         Style["f", Italic]"(3)",
```

```
      Style["f", Italic]"(4)"}
    }
   ]
  },

  {
   Grid[{
     {Row[{"expansion point", Spacer[2], Text@Style["x", Italic]0}]},
     {Manipulator[Dynamic[at, {at = Chop[#];} &],
       {-2, 2, .1}, ImageSize → Tiny], Dynamic@padIt1[at, {3, 2}]
     }}
   ]
  }

 }, Frame → All, FrameStyle -> Directive[Thickness[.005], Gray], Spacings → {0.5, 1.15}
],

Grid[{
  {Text[Style["select accuracy order", 11]], SpanFromLeft},
  {
   Grid[{

     {Dynamic[Style["centered", 12, If[finiteDifferenceType == "centered", Underlined, Black]]],
      SetterBar[Dynamic[orderCentered, {orderCentered = #; orderForward = 0; orderBackward = 0;
           finiteDifferenceType = "centered"; nPoints = If[derivativeOrder == 1 ||
              derivativeOrder == 2, orderCentered + 1, orderCentered + 3]} &], {2, 4}]
     },

     {Dynamic[Style["forward", 12, If[finiteDifferenceType == "forward", Underlined, Black]]],
      SetterBar[Dynamic[orderForward, {orderForward = #; orderCentered = 0; orderBackward = 0;
           finiteDifferenceType = "forward"; nPoints = derivativeOrder + orderForward} &], {1, 2, 3}]
     },
     {Dynamic[Style["backward", 12, If[finiteDifferenceType == "backward", Underlined, Black]]],
      SetterBar[Dynamic[orderBackward, {orderBackward = #; orderForward = 0; orderCentered = 0;
           finiteDifferenceType = "backward"; nPoints = derivativeOrder + orderBackward} &], {1, 2, 3}]
     }

    }, Spacings → {0.1, .4}, Alignment → Left
   ]
  }

 }, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray], Spacings → {0.5, 0.3}
],

Grid[{
  {Style[Row[{"function", Spacer[2], Style["f", Italic], "(", Style["x", Italic], ")"}], 12]},
  {RadioButtonBar[Dynamic[fun], {Cos[#] & → Text@TraditionalForm[Style[Cos[x], 14]],
     Sin[#] & → Text@TraditionalForm[Style[Sin[x], 14]]}, Appearance → "Vertical"]}

 }, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray], Spacings → {0.3, 1.9}
],

Grid[{

  {
   Grid[{
     {Style["plot type", 11]},
     {PopupMenu[Dynamic[plotType, {plotType = #} &],
       {
        1 → Row[{"error at ", Style["x", Italic, FontFamily → "Times"]0}],
        2 → "exact vs. approx."
       }, ImageSize -> {115, 25}
      ]
```

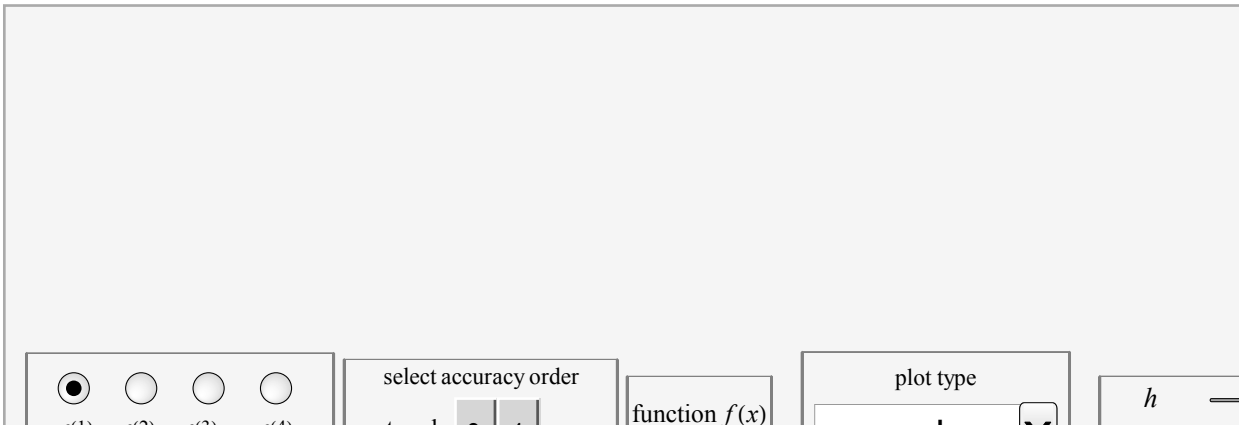
```
    }}, Spacings → {0.4, 0.35}
  ]
 },
 {
  Grid[{
    {Style["select test case", 11]},
    {RadioButtonBar[Dynamic[testCase, {testCase = #,
          Which[testCase == 1,
           derivativeOrder = 1;
           orderForward = 0;
           orderCentered = 0;
           finiteDifferenceType = "backward";
           orderBackward = 1;
           nPoints = derivativeOrder + orderBackward;
           nForApprox = 15;
           n2ForApprox = 1;
           n3ForApprox = 3;
           xForApprox = 3.0;
           yForApprox = 1.2;
           plotType = 2,

           testCase == 2,
           derivativeOrder = 4;
           orderForward = 0;
           orderCentered = 4;
           orderBackward = 0;
           finiteDifferenceType = "centered";
           nPoints =
            If[derivativeOrder == 1 || derivativeOrder == 2, orderCentered + 1, orderCentered + 3];
           nForApprox = 4;
           n2ForApprox = 3;
           n3ForApprox = 0;
           xForApprox = 7.0;
           yForApprox = 1.2;
           plotType = 2,

           testCase == 3,
           derivativeOrder = 3;
           orderForward = 2;
           orderCentered = 0;
           orderBackward = 0;
           finiteDifferenceType = "forward";
           nPoints = derivativeOrder + orderForward;
           nForApprox = 5;
           n2ForApprox = 3;
           n3ForApprox = 0;
           xForApprox = 7.0;
           yForApprox = 1.2;
           plotType = 2,

           testCase == 4,
           derivativeOrder = 1;
           orderForward = 1;
           orderCentered = 0;
           orderBackward = 0;
           finiteDifferenceType = "forward";
           nPoints = derivativeOrder + orderForward;
           nForApprox = 5;
           n2ForApprox = 3;
           n3ForApprox = 0;
           xForApprox = 7.0;
           yForApprox = 1.2;
           plotType = 1
```

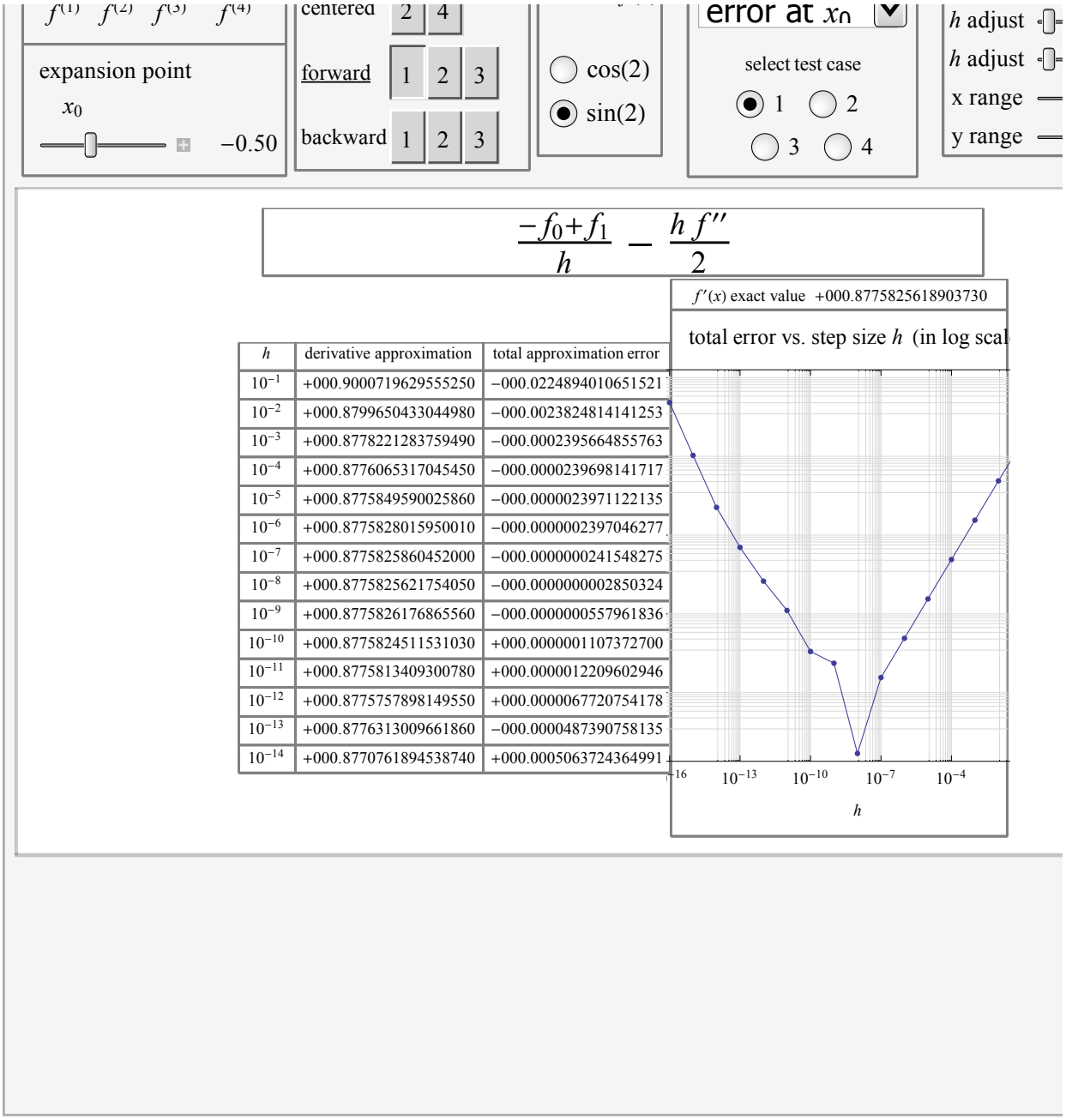
```
            ]
           } &], Range[4]]}
       }, Spacings → {0.4, 0.48}
      ]
     }
    }, Alignment → Center, Spacings → {.8, 0.95},
    Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
   ],

   Grid[{
     {Style["h", Italic], Manipulator[Dynamic[nForApprox, {nForApprox = #} &], {1, 15, 1},
       ImageSize → Tiny, Enabled → Dynamic[plotType == 2]], Dynamic[10^-padIt2[nForApprox,2]]
     },
     {Row[{Style["h", Italic], " adjust"}], Manipulator[Dynamic[n2ForApprox, {n2ForApprox = #} &],
       {1, 9, 1}, ImageSize → Tiny, Enabled → Dynamic[plotType == 2]], Dynamic[padIt2[n2ForApprox, 1]]
     },
     {Row[{Style["h", Italic], " adjust"}], Manipulator[Dynamic[n3ForApprox, {n3ForApprox = #} &],
       {0, 9, 1}, ImageSize → Tiny, Enabled → Dynamic[plotType == 2]], Dynamic[padIt2[n3ForApprox, 1]]
     },
     {"x range", Manipulator[Dynamic[xForApprox, {xForApprox = #} &], {0.1, 7, .1},
       ImageSize → Tiny, Enabled → Dynamic[plotType == 2]], Dynamic[padIt1[xForApprox, {2, 1}]]
     },
     {"y range", Manipulator[Dynamic[yForApprox, {yForApprox = #} &], {0.01, 1.2, .01},
       ImageSize → Tiny, Enabled → Dynamic[plotType == 2]], Dynamic[padIt1[yForApprox, {2, 1}]]
     }
    }, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray], Spacings → {0.5, 0.28}
   ]
  }
 }, Spacings → {0.3, 0.1}, Alignment → Center
],

{{orderCentered, 0}, None},
{{orderForward, 1}, None},
{{orderBackward, 0}, None},

{{xForApprox, 3.0}, None},
{{yForApprox, 1.2}, None},
{{nForApprox, 7}, None},
{{n2ForApprox, 1}, None},
{{n3ForApprox, 0}, None},
{{plotType, 1}, None},
{{fun, Sin[#] &}, None},
{{derivativeOrder, 1}, None},
{{nPoints, 2}, None},
{{at, -0.5}, None},
{{finiteDifferenceType, "forward"}, None},
{{h2, 1}, None},
{{h3, 0}, None},
{{h4, 0}, None},
{{h5, 0}, None},
{{h6, 0}, None},
{{h7, 0}, None},
{{f1, 1}, None},
{{f2, 0}, None},
{{f3, 0}, None},
{{f4, 0}, None},
{{testCase, 1}, None},

SynchronousUpdating → False,
SynchronousInitialization → False,
ContinuousAction → False,
Alignment -> Center,
ImageMargins → 5,
```

```
FrameMargins → 5,
Paneled → True,
Frame → False,
ControlPlacement → Top,
AutorunSequencing → {1},

Initialization :→
 (
  (*definitions used for parameter checking*)
  integerStrictPositive = (IntegerQ[#] && # > 0 &);
  integerPositive = (IntegerQ[#] && # ≥ 0 &);
  numericStrictPositive = (Element[#, Reals] && # > 0 &);
  numericPositive = (Element[#, Reals] && # ≥ 0 &);
  numericStrictNegative = (Element[#, Reals] && # < 0 &);
  numericNegative = (Element[#, Reals] && # ≤ 0 &);
  bool = (Element[#, Booleans] &);
  numeric = (Element[#, Reals] &);
  integer = (Element[#, Integers] &);
  (*-------------------------------------------*)
  padIt1[v_?numeric, f_List] := AccountingForm[v,
    f, NumberSigns → {"-", "+"}, NumberPadding → {"0", "0"}, SignPadding → True];
  (*-------------------------------------------*)
  padIt2[v_?numeric, f_List] := AccountingForm[v,
    f, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}, SignPadding → True];
  (*-------------------------------------------*)
  padIt2[v_?numeric, f_Integer] := AccountingForm[Chop[v],
    f, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}, SignPadding → True];
  (*-------------------------------------------*)

  (*function to generate finite difference formula using interpolating polynomial*)
  mkFormula[order_Integer /; order ≥ 1, (* use 1 to generate approx for f', use 2 for f'' etc...*)
     grid_List?(VectorQ[#, IntegerQ] &),
     (* list of points. For example, for centered difference 3 points use {-1,0,1} etc...*)
     x_, (* symbol to use, has no value*)
     h_, (*symbol to use for spacing, has no value*)
     f_(* symbol to use for f[x] has no value*) ] /; (order ≤ Length[grid] - 1) := Module[{points, e, z, p},

    points = {x + # h, f[x + # h]} & /@ grid;
    p = InterpolatingPolynomial[points, z];
    Simplify[Together[Numerator[q]] / Denominator[q]];
    e = D[p, {z, order}];
    e = e /. z → x;
    Simplify[Together[Numerator[e]] / Denominator[e]]
   ];
  (*-------------------------------------------*)

  (*function to evaluate the finite difference and plot result*)
  mk[order_Integer /; order ≥ 1, (* use 1 to generate approx for f', use 2 for f'' etc...*)
     at_?NumericQ, (*point expansion is made around*)
     grid_List?(VectorQ[#, IntegerQ] &),
     (* list of points. For example, for centered difference 3 points use {-1,0,1} etc...*)
     fun_, (* pure function using # for x. This is the function to approximate its derivative*)
     f_, (* symbol to use for f[x] has no value*)
     h_, (* symbol to use for h has no value*)
     x_, (* symbol to use for x has no value*)
     plotType_Integer?Positive,
     nForApprox_Integer?Positive,
     n2ForApprox_Integer?Positive,
     n3ForApprox_Integer?NonNegative,
     xForApprox_Real?Positive,
     yForApprox_Real?Positive,
```

```
  finiteDifferenceType_String] /; order ≤ (Length[grid] - 1) :=
Module[{expr, hValues, e, n, appr, exact, parms, data, i, lte, hOrder = 16, hValueUsed},

 parms = {f[x] -> f0, f[x - 2 h] -> f-2, f[x - h] -> f-1, f[x + h] -> f1, f[x + 2 h] -> f2, f[x + 3 h] -> f3, f[x + 4 h] ->
    f4, f[x - 3 h] -> f-3, f[x - 4 h] -> f-4, f[x - 5 h] -> f-5, f[x - 6 h] -> f-6, f[x + 5 h] -> f5, f[x + 6 h] -> f6};
 (*Use for final display of finite difference formula to make it smaller to fit*)

 hValues = Table[1. / (10^n), {n, 1, hOrder}];
 e = mkFormula[order, grid, x, h, f]; (*this generated the difference formula*)
 appr = e /. {f → fun, x → at, h → #} & /@ hValues;
 exact = D[fun[x], {x, order}] /. x → at;

 expr = TraditionalForm[D[f[x], {x, order}]];
 lte = Last@FDFormula[order, Length[grid] - 1,
    Which[finiteDifferenceType == "centered", (Length[grid] - 1) / 2,
     finiteDifferenceType == "forward", 0,
     True, Length[grid] - 1]
   ];
 (*reverse the sign to move it to other side*)
 lte = -lte;

 {(*build return list *)
  lte,
  expr,
  e /. parms,

  Grid[Transpose[{
     Flatten@{Style["h", Italic], HoldForm[10^-#] & /@ Range[14] },
     Flatten[{"derivative approximation",
       If[Abs[#] > 999, "*", padIt1[#, {19, 16}]] & /@ appr[[1 ;; 14]]}],

     Flatten@{Row[{"total approximation error"}],
       If[Abs[#] > 999, "*", padIt1[#, {19, 16}]] & /@ (exact - appr)[[1 ;; 14]]}
    }], Alignment → Center, Frame → All,
   FrameStyle -> Directive[Thickness[.005], Gray], Spacings → {1, .5}
  ],

  Item[Row[{expr, " exact value", Spacer[5], padIt1[exact, {19, 16}]}], ItemSize → {23, 1.8}],
  Which[plotType == 1,

   ListLogLogPlot[Transpose@{hValues, Abs[exact - appr]},
    Frame → True, Joined → True, Mesh → All, FrameLabel → {{None, None},
      {Style["h", Italic], Style[Grid[{
          {Text@Row[{"total error vs. step size ", Style["h", Italic], Spacer[5], "(in log scale)"}]}
         }, Spacings → {.5, .4}, Alignment → Center
        ], 12]
      }},
    RotateLabel → True,
    GridLines → Automatic, GridLinesStyle → LightGray,
    ImageSize → {261, 271},
    ImagePadding → {{30, 10}, {32, 25}},
    ImageMargins → 8,
    AspectRatio → 1],

   plotType == 2,
   hValueUsed = N[(n2ForApprox * 10^(-nForApprox) + n3ForApprox * 10^(-nForApprox - 1))];
   data =
    Table[{i, e /. {f → fun, x → i, h → hValueUsed}}, {i, -xForApprox, xForApprox, xForApprox / 100.}];

   Show[
    Plot[Evaluate@D[fun[x], {x, order}], {x, -xForApprox, xForApprox},
     Frame → True,
     GridLines → Automatic, GridLinesStyle → LightGray,
```

```
        ImagePadding → {{20, 5}, {32, 45}},
        ImageMargins → 8,
        AspectRatio → 1,
        ImageSize → {261, 271},
        Axes → False,
        PlotRange → {All, {-yForApprox, yForApprox}},
        FrameLabel → {{None, None}, {Style["x", Italic],
           Style[Grid[{
              {Row[{"exact vs. approximate ", expr}]},
              {Row[{Style["h", Italic], " = ", padIt2[hValueUsed, {16, 16}]}]}
             }], 12]
          }}
       ]
       ,
       ListLinePlot[data, Joined → True, Mesh → True, PlotStyle → Red, ImagePadding → 0]
      ]
     ]
    }
   ];
(*-----------------------------*)
(*function to generate points for expansion*)
makePoints[nPoints_Integer /; nPoints ≥ 2, finiteDifferenceType_String] := Module[{d = (nPoints - 1) / 2},
  Which[finiteDifferenceType == "centered", Table[n, {n, -d, d, 1}],
   finiteDifferenceType == "forward", Table[n, {n, 0, nPoints - 1, 1}],
   finiteDifferenceType == "backward", Table[n, {n, -nPoints + 1, 0, 1}]
  ]
 ];
(*-----------------------------*)
(*FDFormula from http://reference.wolfram.com/mathematica/tutorial/NDSolvePDE.html*)
FDFormula[m_Integer?Positive(*order of derivative*), n_Integer?Positive(*number of grid intervals*),
  s_Integer?NonNegative(*set to n/2 to get centered differenrer*)] := Module[{},

  F = Table[f[Subscript[x, i + k]], {k, -s, n - s}];
  W = PadRight[CoefficientList[Normal[Series[x^s Log[x]^m, {x, 1, n}] / h^m], x], Length[F], 0];
  Wfact = 1 / Apply[PolynomialGCD, W];
  W = Simplify[W Wfact];
  taylor[h_] = Normal[Series[f[Subscript[x, i] + h], {h, 0, n + 2}]];
  error = Drop[CoefficientList[Expand[(Table[taylor[h k], {k, -s, n - s}].W) / Wfact], h], 1];
  do = Position[error, e_ /; e ≠ 0][[1, 1]];
  error = error[[do]];
  error = error /. f_[Subscript[x, i]] → f;
  error = h^do error;

  {Derivative[m][f][Subscript[x, i]] ≈ (F.W) / Wfact, error}
 ]
)]
```





h	derivative approximation	total approximation error
10^{-1}	+000.9000719629555250	−000.0224894010651521
10^{-2}	+000.8799650433044980	−000.0023824814141253
10^{-3}	+000.8778221283759490	−000.0002395664855763
10^{-4}	+000.8776065317045450	−000.0000239698141717
10^{-5}	+000.8775849590025860	−000.0000023971122135
10^{-6}	+000.8775828015950010	−000.0000002397046277
10^{-7}	+000.8775825860452000	−000.0000000241548275
10^{-8}	+000.8775825621754050	−000.0000000002850324
10^{-9}	+000.8775826176865560	−000.0000000557961836
10^{-10}	+000.8775824511531030	+000.0000001107372700
10^{-11}	+000.8775813409300780	+000.0000012209602946
10^{-12}	+000.8775757898149550	+000.0000067720754178
10^{-13}	+000.8776313009661860	−000.0000487390758135
10^{-14}	+000.8770761894538740	+000.0005063724364991

Caption

This Demonstration illustrates the effect of numerical errors on the approximation of derivatives when using the finite-difference scheme with different step sizes and different orders of accuracy. You can select to approximate up to the fourth derivative, the desired local truncation accuracy order $O(h^n)$, and the finite difference scheme to use (centered, forward, or backward).

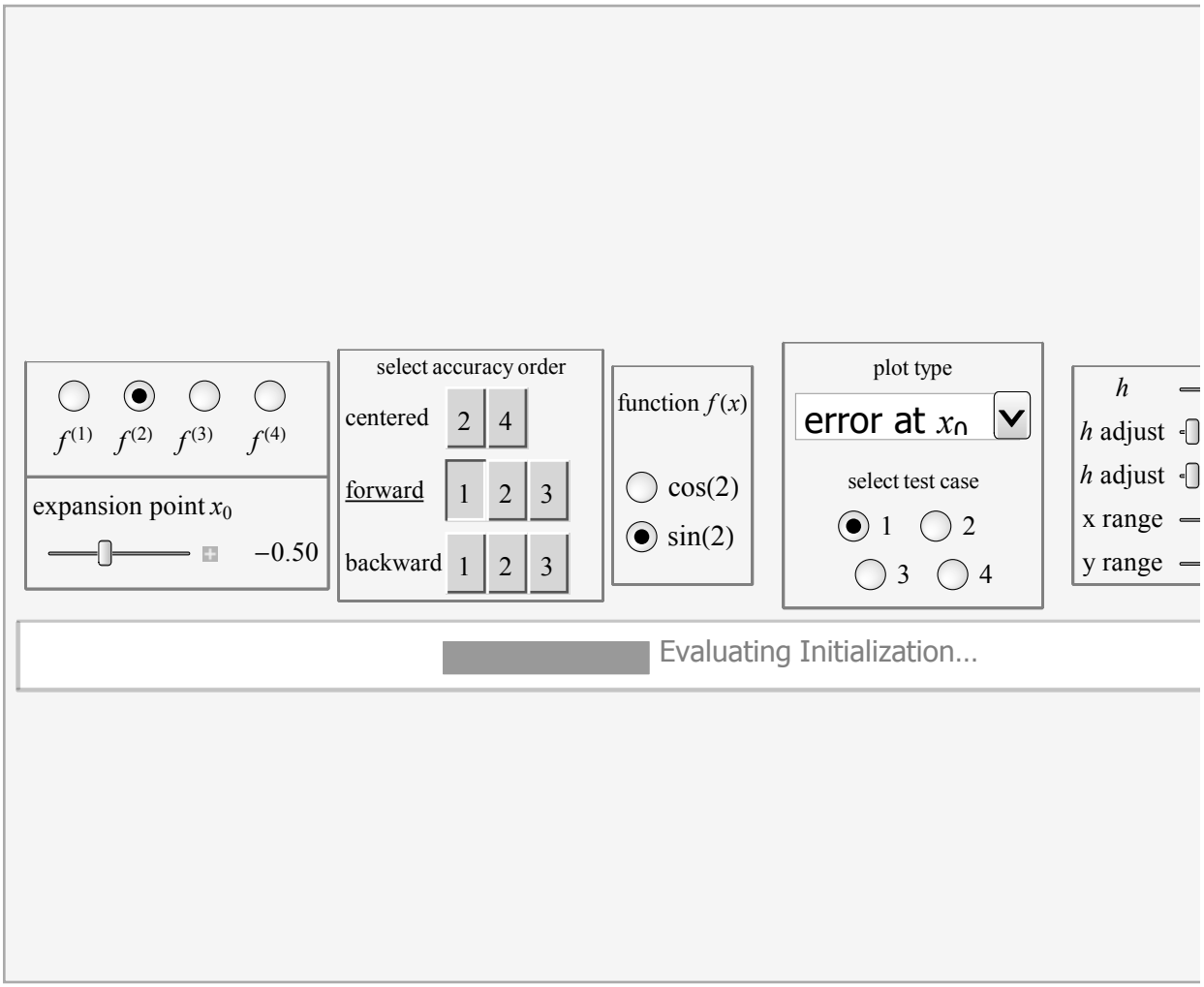
The total computational error is made up of two components:

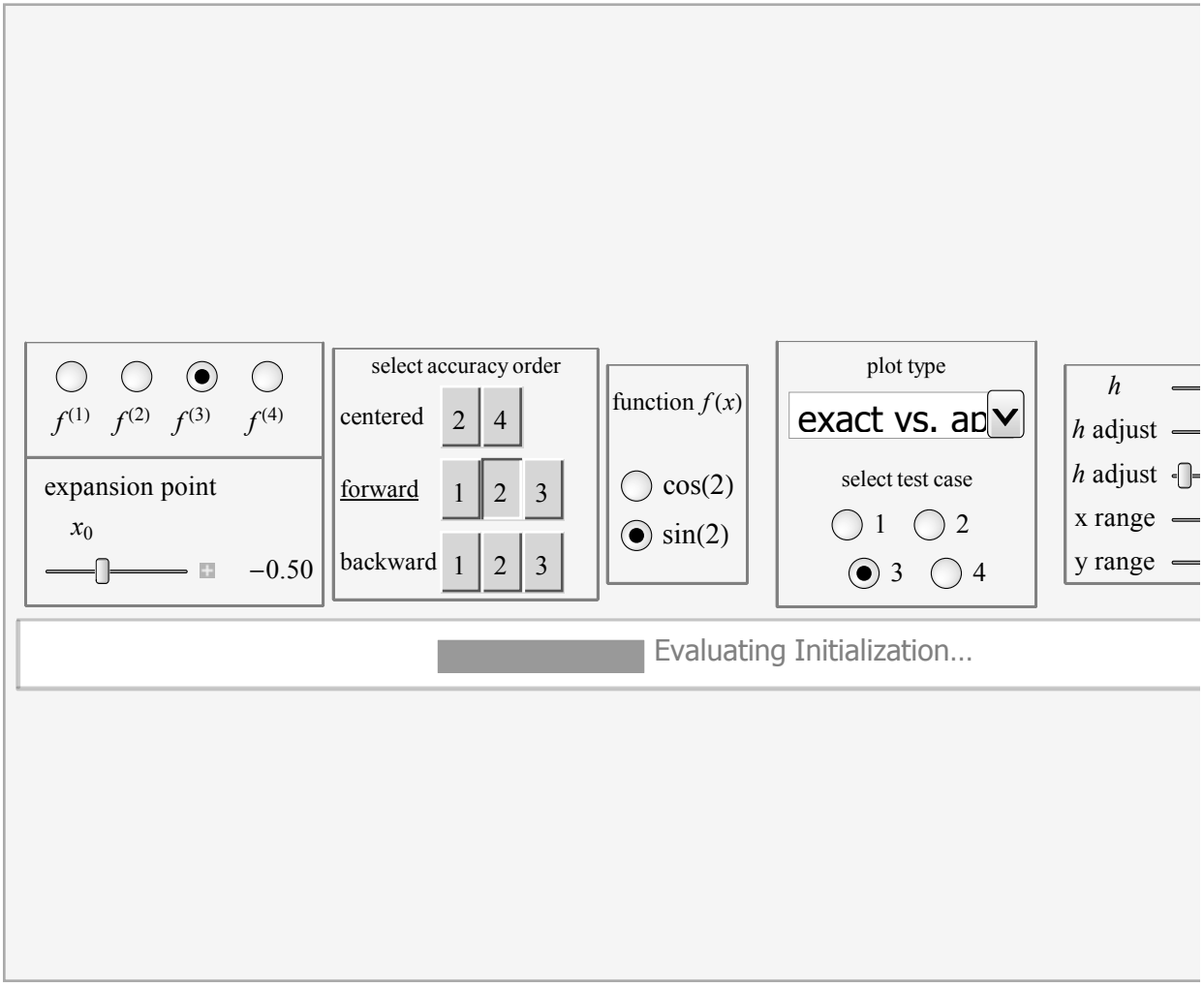
1. The local truncation error (LTE, also called the algorithmic error) is due to dropping the higher order terms in the Taylor series expansion.
2. Rounding errors are due to floating-point arithmetic.

Reducing the LTE is achieved by reducing the step size h. Since the rounding error is inversely proportional to the step size, this in turn causes the rounding error to increase. When the step size becomes smaller than an optimal value, the total error itself start to increase. This is called the step-size dilemma: Reducing truncation error increases rounding error. There exists an optimal step size h

for which the total error is minimum. This optimal step size can be found analytically by minimizing the total error function with respect to the step size h. This optimal step size is a function of the expansion point x, the finite difference scheme used, and the order of the scheme. Using a higher order gives a more accurate result at the optimal step size. In addition, the optimal step size becomes larger in order to avoid rounding errors. On the other hand, using higher order requires more computations, as the finite-difference scheme contains more terms.

Thumbnail





Details (optional)

The total error in approximating a derivative using the finite difference method is made up of local truncation error (LTE) and rounding error. The rounding error is proportional to ϵ/h, where ϵ is the machine epsilon and h is the step size. LTE is directly proportional to $O(h^n)$ where n is the order of approximation. This is called the big O order of the scheme. This is the right-most term in the finite difference formula displayed. As h becomes smaller, rounding errors increase and the LTE decreases. An optimal step size h gives the minimum total error in approximation.

A log scaled plot shows this optimal h value. A typical plot shows that the total error decreases initially as h becomes smaller, because the truncation error is decreasing while the rounding error remains relatively small in proportion. However, the total error starts increasing when h becomes smaller than the optimal value. This is the point at which the rounding error starts to grow in larger proportion than the decrease in LTE can compensate.

The finite-difference formula is found by using derivatives of an interpolating polynomial. To find the finite-difference formula using m grid points, a polynomial of order $m-1$ is differentiated. This method is recommended over using Taylor series expansion when the number of grid points becomes large. The Fornberg formula can also be used to generate the finite-difference formula efficiently for larger numbers of grid points. The Fornberg formula is used in this Demonstration to obtain the truncated term in Taylor series (order of scheme) while the polynomial interpolation method is used to generate the finite-difference scheme in order to illustrate the method.

Another plot is generated that compares the exact derivative to the approximated derivative over the whole range. You select the function and the degree of the derivative to approximate and the location of the expansion point x.

You can select up to the fourth derivative, the desired order of the scheme, and the type of the finite difference method (centered, forward, or backward). This Demonstration automatically determines the minimum required number of grid point to obtain this order, and the finite-difference formula is displayed at the top, including the leading truncated error term in the Taylor series, which is the right-most term.

A table showing the step size h and the corresponding value of the approximated derivative and the total error at the point of

expansion is also given. The exact value of the derivative being approximated at the point of expansion is displayed on the right side of the display in the title of the plot.
When the error is larger than 1000 a star "*" is shown in the table for that step size instead of the actual error value. You can obtain finer control on the step size h value using the additional two sliders below the slider labeled "h". You can select from a number of test cases.
References

[1] Wikipedia page on finite difference coefficient.

[2] M. Fogiel, *REA's Problem Solver, Numerical Analysis*, New Jersey: REA, 1993.

[3] B. Carnahan, H. Luther, and J. Wilkes, *Applied Numerical Methods*, New York: John Wiley, 1969.

Control Suggestions (optional)

- [x] Resize Images
- [ ] Rotate and Zoom in 3D
- [ ] Drag Locators
- [ ] Create and Delete Locators
- [x] Slider Zoom
- [x] Gamepad Controls
- [x] Automatic Animation
- [ ] Bookmark Animation

Search Terms (optional)

Fornberg formula
finite difference method
Taylor series
LTE local truncation error
step size dilemma
forward
backward
centered

Related Links (optional)

finite difference
Numerical Solution of Partial Differential Equations

Authoring Information

Contributed by: Nasser M. Abbasi