

Finite Elements - 1D elliptic problem

Taken from R.J.LeVeque: Finite Difference Methods for Ordinary and Partial Differential Equations - Steady-State and Time-Dependent Problems, SIAM, Philadelphia 2007, chapter 4.6

Preliminaries

Clear all symbols from previous evaluations to avoid conflicts

```
In[1]:= Clear ["Global`*"]
```

Problem

Differential equation

We would like to solve numerically the differential equation

$$\frac{d^2 u(x)}{dx^2} = f(x) \quad (1)$$

with Dirichlet boundary conditions

$$\begin{aligned} u(a) &= u_a \\ u(b) &= u_b \end{aligned} \quad (2)$$

Particular problem

As the right-hand-side we will take

$$f(x) = -20 + c \phi''(x) \cos(\phi(x)) - c(\phi'(x))^2 \sin(\phi(x)) \quad (3)$$

where

$$c = 1/2$$

$$\phi(x) = 20 \pi x^3$$

and boundary conditions are

$$\begin{aligned} u(0) &= 1 \\ u(1) &= 3 \end{aligned} \quad (4)$$

The analytic solution

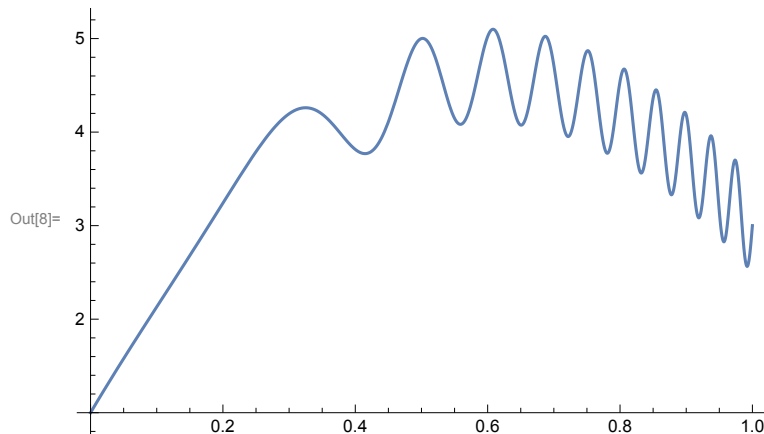
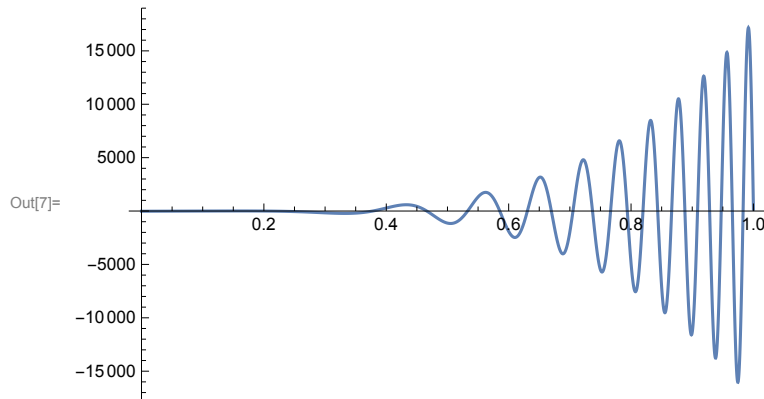
This problem can be solved in the closed form

```

In[2]:= a = 0; b = 1; u_a = 1; u_b = 3; c = 1/2;
        ϕ[x_] = 20 π x^3;
        f[x_] = -20 + c ∂x,x ϕ[x] Cos[ϕ[x]] - c (∂x ϕ[x])^2 Sin[ϕ[x]];
        (* Delete[... , 0] deletes outer {} of a list *)
        sol = Delete[DSolve[{∂x,x U[x] == f[x], U[a] == u_a, U[b] == u_b}, U[x], x], 0];
        u[x_] = Expand[U[x] /. sol]
        Plot[f[x], {x, a, b}, PlotRange → All]
        Plot[u[x], {x, a, b}, PlotRange → All]

```

Out[6]= $1 + 12x - 10x^2 + \frac{1}{2} \sin[20\pi x^3]$



Direct numerical solution - finite difference

We discretize the equation (1) using a standard finite difference formula on the equidistant grid

$x_j = jh$ where $h = 1/(n+1)$, $j = 0, \dots, n+1$

i.e. we have to solve the system of n equations

$$\frac{d^2 u(x_j)}{dx^2} \approx \frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1}))}{h^2} = f(x_j) \quad \text{for } j = 1, \dots, n \quad (5)$$

with boundary conditions

$$u(x_0 = 0) = 1, \quad u(x_{n+1} = 1) = 3$$

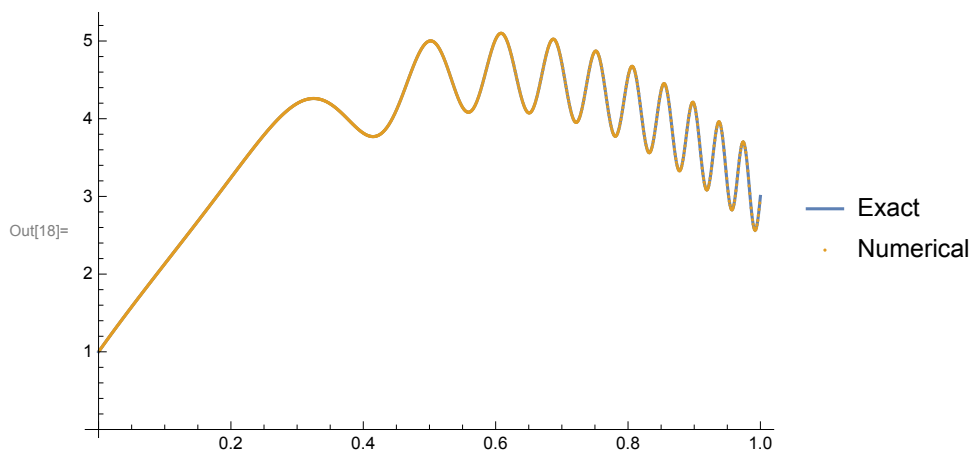
We can use *Mathematica* to solve this problem directly. For later convenience in gradient methods we actually solve

$$-\frac{d^2 u(x_j)}{dx^2} \simeq \frac{-u(x_{j+1}) + 2u(x_j) - u(x_{j-1}))}{h^2} = -f(x_j) \text{ for } j = 1, \dots, n$$

```

In[9]:= n = 1000; h = (b - a) / (n + 1);
X = Range[a, b, h]; Xin = X[[2 ;; n + 1]];
T = SparseArray[{{i_, i_} -> 2.0, {i_, j_} /; Abs[i - j] == 1 -> -1.0}, {n, n}];
rhs = -N[h^2 f[Xin], 16];
rhs[[1]] = rhs[[1]] + u_a; (* boundary condtions *)
rhs[[n]] = rhs[[n]] + u_b;
xDirect = LinearSolve[T, rhs];
Print["MaxError = ", Max[Abs[u[X[[2 ;; n + 1]]] - xDirect]]]
exactSol = Transpose[{X, u[X]}]; (* for later use in plots *)
ListPlot[{exactSol, Transpose[{Xin, xDirect}]},
  Joined -> {True, False}, PlotLegends -> {"Exact", "Numerical"}]
MaxError = 0.001414579756

```



Finite elements solution with DVR basis in each element

Gauss-Lobatto quadrature points and weights will be used for integration over one element

```

In[19]:= getGaussLobattoPointsAndWeights[n_, a_, b_] :=
Module[{x, w, p},
  (* roots of the derivative of the (n-1)st Legendre polynomial are inner points
    of the Gauss-Lobatto quadrature on [-1,1]*)
  p[z_] = LegendreP[n - 1, z];
  If[n == 2,
    x = {-1.0, 1.0},
    x = N[Flatten[{-1.0, Sort[Re[z /. N[Solve[D[p[z], z] == 0, z], 16]]], 1.0}]]
  ];
  (* to get weights we need values of this polynomial *)
  w = 2.0
  Flatten[{1.0, Table[1.0 / (N[p[x[[i]]], 16)]^2, {i, 2, n - 1}], 1.0}] / (n (n - 1));
  (* shifting and scaling to the interval [a,b] *)
  x = (b - a) x / 2 + (b + a) / 2;
  w = (b - a) w / 2;
  Return[{x, w}]
]
getGaussLobattoPointsAndWeights[5, 0, 1]

```

```

Out[20]= {{0., 0.1726731646, 0.5, 0.8273268354, 1.},
  {0.05, 0.2722222222, 0.3555555556, 0.2722222222, 0.05}}

```

Full grid and weights

```

In[21]:= getFEMPointsAndWeights[nGL_, endpoints_] :=
Module[{nEl, nPoints, xGL, wGL, x, w},
  nEl = Length[endpoints] - 1;
  nPoints = nEl * (nGL - 1) + 1;
  (* Print["Number of all points/basis functions is ", nPoints]; *)
  x = ConstantArray[0.0, nPoints];
  w = ConstantArray[0.0, nPoints];
  Do[
    {xGL, wGL} =
      getGaussLobattoPointsAndWeights[nGL, endpoints[[i]], endpoints[[i + 1]]];
    x[[ (i - 1) * (nGL - 1) + 1 ;; i * (nGL - 1) + 1]] = xGL;
    (* weights at points which are common to two elements are added up *)
    w[[ (i - 1) * (nGL - 1) + 1 ;; i * (nGL - 1) + 1]] += wGL,
    {i, 1, nEl}
  ];
  Return[{x, w}]
]
getFEMPointsAndWeights[4, {0, 1, 3, 6}]

```

```

Out[22]= {{0., 0.2763932023, 0.7236067977, 1., 1.552786405, 2.447213595, 3.,
  3.829179607, 5.170820393, 6.}, {0.08333333333, 0.4166666667, 0.4166666667,
  0.25, 0.8333333333, 0.8333333333, 0.4166666667, 1.25, 1.25, 0.25}}

```

Derivatives of the Lagrange polynomials at GL points on [-1,1] - result is a matrix nGL x nGL of $D[l_i(x), x = x_k]$

```

In[23]:= derivativesLagPol[nGL_] :=
Module[{xGL, wGL, dLP, hlp},
  dLP = ConstantArray[0.0, {nGL, nGL}];
  {xGL, wGL} = getGaussLobattoPointsAndWeights[nGL, -1.0, 1.0];
  Do[
    (* Diagonal terms *)
    dLP[[i, i]] = 0.0;
    Do[
      If[i ≠ s, dLP[[i, i]] = dLP[[i, i]] + 1.0 / (xGL[[i]] - xGL[[s]]),
      {s, 1, nGL}
    ];
    (* Off-diagonal terms *)
    Do[
      hlp = 1.0;
      Do[
        If[(j ≠ i) && (j ≠ k), hlp = hlp * (xGL[[k]] - xGL[[j]]) / (xGL[[i]] - xGL[[j]]),
        {j, 1, nGL}
      ];
      dLP[[i, k]] = hlp / (xGL[[i]] - xGL[[k]]);
      dLP[[k, i]] = 1.0 / (hlp * (xGL[[k]] - xGL[[i]])),
      {k, i + 1, nGL}
    ],
    {i, 1, nGL}
  ];
  Return[dLP];
];
derivativesLagPol[4]

```

```

Out[24]= {{-3., -0.8090169944, 0.3090169944, -0.5},
{4.045084972, -3.330669074 × 10-16, -1.118033989, 1.545084972},
{-1.545084972, 1.118033989, 2.220446049 × 10-16, -4.045084972},
{0.5, -0.3090169944, 0.8090169944, 3.}}

```

Construction of the stiffness matrix (ϕ'_i, ϕ'_j)

```

In[25]:= constructStiffnessMatrix[nGL_, endPoints_] :=
Module[{nEl, nPoints, xFEM, wFEM, xGL, wGL, dLP, dBF, k1, ii, jj, oldCorner, A},
  nEl = Length[endPoints] - 1;
  nPoints = nEl * (nGL - 1) + 1;
  (* get weights for all points *)
  {xFEM, wFEM} = getFEMPointsAndWeights[nGL, endPoints];
  (* calculate derivatives of the Lagrange
  interpolating polynomials at GL points on [-1,1] *)
  dLP = derivativesLagPol[nGL];
  (* build the stiffness matrix *)
  A = ConstantArray[0.0, {nPoints, nPoints}];
  oldCorner = 0.0;
  Do[
    {xGL, wGL} =
      getGaussLobattoPointsAndWeights[nGL, endPoints[[k]], endPoints[[k + 1]]];
    (* dilatation of derivatives of LP to be the derivatives
    of the basis functions on the k-th element *)
    dBF = 2.0 * dLP / (endPoints[[k + 1]] - endPoints[[k]]);
    k1 = (k - 1) * (nGL - 1) + 1;
    (* index of the first point of the k-th element in x *)
    Do[
      (* normalization factor of basis functions *)
      dBF[[i, All]] = dBF[[i, All]] / Sqrt[wFEM[[k1 + i - 1]]],
      {i, 1, nGL}
    ];
    Do[
      ii = k1 + i - 1; (* current row in the A matrix *)
      Do[
        jj = k1 + j - 1; (* current column in the A matrix *)
        A[[ii, jj]] = -Sum[wGL[[s]] * dBF[[i, s]] * dBF[[j, s]], {s, 1, nGL}];
        A[[jj, ii]] = A[[ii, jj]],
        {j, i, nGL}
      ],
      {i, 1, nGL}
    ];
    A[[k1, k1]] += oldCorner;
    oldCorner = A[[k1 + nGL - 1, k1 + nGL - 1]],
    {k, 1, nEl}
  ];
  Return[A]
]
constructStiffnessMatrix[2, {0, 1, 2, 3, 4}] // MatrixForm

```

Out[26]//MatrixForm=

$$\begin{pmatrix} -2. & 1.414213562 & 0. & 0. & 0. \\ 1.414213562 & -2. & 1. & 0. & 0. \\ 0. & 1. & -2. & 1. & 0. \\ 0. & 0. & 1. & -2. & 1.414213562 \\ 0. & 0. & 0. & 1.414213562 & -2. \end{pmatrix}$$

Simple test using elements of the same length:

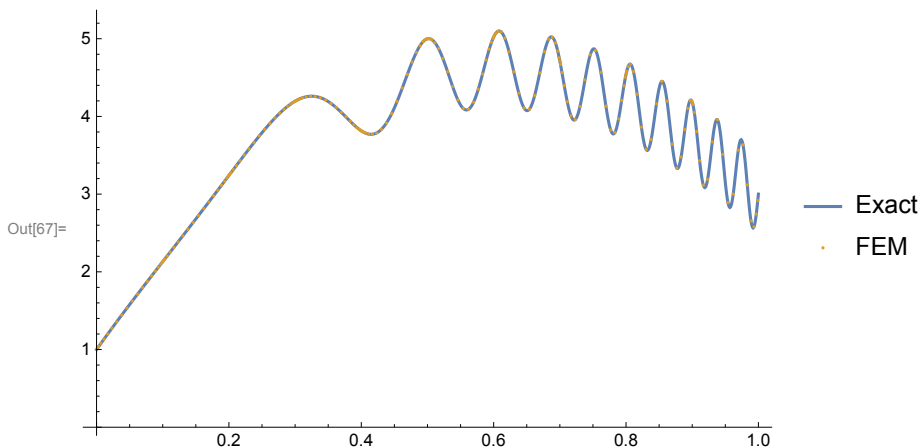
```

In[55]:= nGL = 20;
nEl = 10;
endPoints = Table[N[a + i * (b - a) / nEl], {i, 0, nEl}];
{xFEM, wFEM} = getFEMPointsAndWeights[nGL, endPoints];
Nb = Length[xFEM];
Print["Number of points/basis functions: ", Nb];
(* coefficients of the right-hand-side function f(x) in the FEM basis *)
vf = N[f[xFEM]] * Sqrt[wFEM];
A = constructStiffnessMatrix[nGL, endPoints];
(* right-hand side of the system of linear
   equations must be modified due to boundary conditions *)
bin = vf[[2 ;; Nb - 1]] - u_a * Sqrt[wFEM[[1]]] * A[[2 ;; Nb - 1, 1]] -
      u_b * Sqrt[wFEM[[Nb]]] * A[[2 ;; Nb - 1, Nb]];
(* we actually need only "inner part" (without the first and last rows and columns)
   of the stiffness matrix A *)
Ain = A[[2 ;; Nb - 1, 2 ;; Nb - 1]];
(* to get the functional values of the solution at grid
   points we have to multiply the coefficients by Sqrt[w] *)
uFEM = LinearSolve[Ain, bin] / Sqrt[wFEM[[2 ;; Nb - 1]]];
(* comparison with the exact solution *)
Print["MaxError: ", Max[Abs[u[xFEM[[2 ;; Nb - 1]]] - uFEM]];
exactSol = Transpose[{X, u[X]}]; (* for later use in plots *)
ListPlot[{exactSol, Transpose[{xFEM[[2 ;; Nb - 1]], uFEM]}],
  Joined -> {True, False}, PlotLegends -> {"Exact", "FEM"}]

```

Number of points/basis functions: 191

MaxError: $3.134492266 \times 10^{-8}$



Simple test using elements of the various lengths:

```

In[41]:= nGL = 20;
nEl = 10;
endPoints = {0.0, 0.3, 0.5, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0};
{xFEM, wFEM} = getFEMPointsAndWeights[nGL, endPoints];
Nb = Length[xFEM];
Print["Number of points/basis functions: ", Nb];
(* coefficients of the right-hand-side function f(x) in the FEM basis *)
vf = N[f[xFEM]] * Sqrt[wFEM];
A = constructStiffnessMatrix[nGL, endPoints];
(* right-hand side of the system of linear
   equations must be modified due to boundary conditions *)
bin = vf[[2 ;; Nb - 1]] - u_a * Sqrt[wFEM[[1]]] * A[[2 ;; Nb - 1, 1]] -
      u_b * Sqrt[wFEM[[Nb]]] * A[[2 ;; Nb - 1, Nb]];
(* we actually need only "inner part" (without the first and last rows and columns)
   of the stiffness matrix A *)
Ain = A[[2 ;; Nb - 1, 2 ;; Nb - 1]];
(* to get the functional values of the solution at grid
   points we have to multiply the coefficients by Sqrt[w] *)
uFEM = LinearSolve[Ain, bin] / Sqrt[wFEM[[2 ;; Nb - 1]]];
(* comparison with the exact solution *)
Print["MaxError = ", Max[Abs[u[xFEM[[2 ;; Nb - 1]]] - uFEM]]]
exactSol = Transpose[{X, u[X]}]; (* for later use in plots *)
ListPlot[{exactSol, Transpose[{xFEM[[2 ;; Nb - 1]], uFEM]}],
  Joined -> {True, False}, PlotLegends -> {"Exact", "FEM"}]

```

Number of points/basis functions: 191

MaxError = $3.231974688 \times 10^{-10}$

