Multigrid method

1D elliptic problem

Clear all symbols from previous evaluations to avoid problems

```
Clear["Global`*"]
```

Problem

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

Differential equation

We would like to solve numerically the differential equation

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

with Dirichlet boundary conditions

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

Particular problem

As the right-hand-side we will take

$$f(x) = -20 + c\,\phi''(x)\cos(\phi(x)) - c(\phi'(x))^2 \sin(\phi(x)) \tag{3}$$

where

$$c = 1/2$$

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

and boundary conditions are

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

The analytic solution

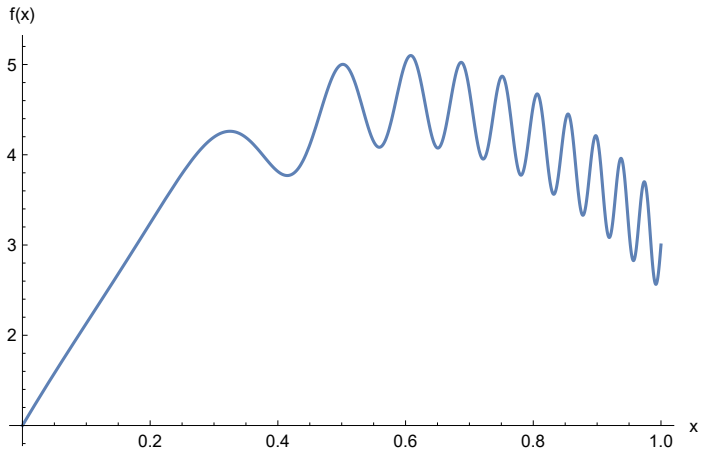
This problem can be solved in the closed form

In[62]:=
```
a = 0; b = 1; ua = 1; ub = 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]];
sol = Delete[DSolve[{∂x,x U[x] == f[x], U[a] == ua, U[b] == ub}, U[x], x], 0];
(* Delete[...,0] deletes outer {} *)
u[x_] = Expand[U[x] /. sol]
Plot[u[x], {x, a, b}, AxesLabel → {"x", "f(x)"}]
```

Out[66]=

$$1 + 12\,x - 10\,x^2 + \frac{1}{2}\,\text{Sin}\left[20\,\pi\,x^3\right]$$

Out[67]=



Direct numerical solution

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

$x_j = a + j\,h$ where $h = (b-a)/(n+1)$, $j = 0, \ldots, n+1$

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

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

with boundary conditions

$u(x_0 = a) = u_a$, $u(x_{n+1} = b) = u_b$

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

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

Evaluating the function $f(x)$ for many x_j takes a long time in *Mathematica* therefore we calculate it in advance and use the vector *fx* in all calculations later. We also calculate the exact solution on the grid as *ux* for later use.

In[7]:=
```
ig = 10; (* the highest level used in multigrid below *)
n = 2^ig - 1; h = (b - a) / (n + 1);
Print["Highest number of points: n = ", n]
X = Range[a, b, h];
Xin = X[[2 ;; n + 1]];
(* fx will be a packed array (machine precission) thanks to N[] *)
fx = N[f[Xin]];
ux = N[u[X]];
```

```
Highest number of points: n = 1023
```

The right-hand-side will be $-h^2 f(x_j)$ but the first and last element must be modified using Dirichlet's boundary conditions $u(x_0)$ for

$$-\frac{d^2 u(x_1)}{dx^2} \simeq \frac{-u(x_0) + 2\,u(x_1) - u(x_2)}{h^2} = -f(x_1) \Longrightarrow 2\,u(x_1) - u(x_2) = -h^2 f(x_1) + u_a$$

$$-\frac{d^2 u(x_n)}{dx^2} \simeq \frac{-u(x_{n-1}) + 2\,u(x_n) - u(x_{n+1})}{h^2} = -f(x_n) \Longrightarrow -u(x_{n-1}) + 2\,u(x_n) = -h^2 f(x_n) + u_b$$

In[14]:=
```
rhs = -h^2 fx;
rhs[[1]] = rhs[[1]] + ua;
rhs[[n]] = rhs[[n]] + ub;
```

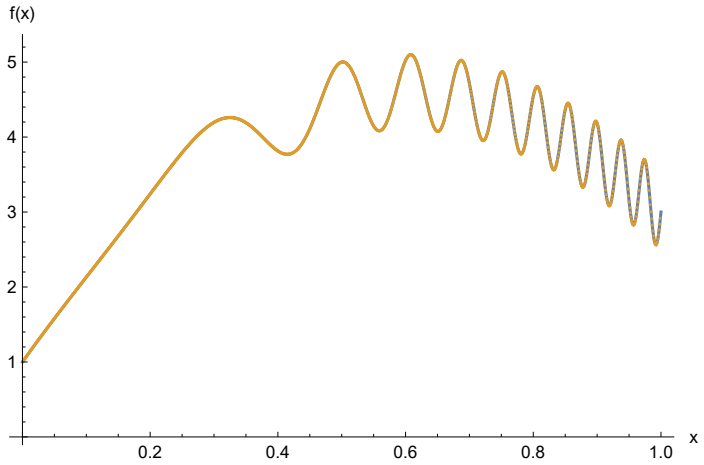
Now we are ready to solve the problem on the grid:

In[17]:=
```
T = SparseArray[{{i_, i_} → 2.0, {i_, j_} /; Abs[i - j] == 1 → -1.0}, {n, n}];
xDirect = LinearSolve[T, rhs];
```

In[68]:=
```
Print["Maximum error = ", Max[Abs[ux[[2 ;; n + 1]] - xDirect]]]
exactSol = Transpose[{X, ux}]; (* for later use in plots *)
ListPlot[{exactSol, Transpose[{Xin, xDirect}]},
 Joined → {True, False}, AxesLabel → {"x", "f(x)"}]
```

```
Maximum error = 0.00134889730827
```

Out[70]=



Multigrid solution - using only the same V-cycle

Multigrid V-cycle algorithm (MGV) starting on the level (i) consists of these steps:

1. improving the solution (here using the weighted Jacobi method with $\omega = 2/3$)

$$x^{(i)} = \text{Smoothing}\left(b^{(i)}, x^{(i)}\right) \tag{6}$$

2. computing the residual

$$r^{(i)} = T^{(i)}\, x^{(i)} - b^{(i)} \tag{7}$$

3. solving the problem

$$T^{(i)}\, d^{(i)} = r^{(i)} \tag{8}$$

by recursive restriction on the coarser grids (4 is there because for the coarser grid we get $(2\,h)^2$ on the right-hand-side)

$$d^{(i)} = \text{Interpolation}\left(\text{MGV}\left(4\ \text{Restriction}\left(r^{(i)}\right), 0\right)\right) \tag{9}$$

4. correcting the fine grid solution

$$x^{(i)} = x^{(i)} - d^{(i)} \tag{10}$$

5. improving the solution again (usually using the weighted Jacobi method with $\omega = 2/3$)

$$x^{(i)} = \text{Smoothing}\left(b^{(i)}, x^{(i)}\right) \tag{11}$$

If (i) is the lowest level (1) then we solve the problem directly.

The following functions do the particular jobs described above:

In[22]:=
```
smoothing[b_, x0_, ω_, niter_] :=
 Module[{n, xNew, R, c, ω2},
  n = Length[x0];
  xNew = x0;
  ω2 = 0.5 ω;
  R = SparseArray[{{i_, i_} → (1.0 - ω), {i_, j_} /; Abs[i - j] == 1 → ω2}, {n, n}];
  c = ω2 b;
  Do[
   xNew = R.xNew + c,
   {i, 1, niter}
  ];
  Return[xNew]
 ]
restriction[r_] :=
 Module[{n, P},
  n = Length[r];
  P = SparseArray[{{i_, j_} /; 2 i - 1 == j → 0.25,
     {i_, j_} /; 2 i == j → 0.5, {i_, j_} /; 2 i + 1 == j → 0.25}, {(n - 1) / 2, n}];
  Return[P.r]
 ]
interpolation[r_] :=
 Module[{n, P},
  n = Length[r];
  P = SparseArray[{{i_, j_} /; 2 j - 1 == i → 0.5,
     {i_, j_} /; 2 j == i → 1.0, {i_, j_} /; 2 j + 1 == i → 0.5}, {2 n + 1, n}];
  Return[P.r]
 ]
MGV[b_, u_] :=
 Module[{n, uNew, T, r, d},
  n = Length[b];
  If[n == 1,
   (* direct solution *)
   uNew = 0.5 b,
   (* iterations *)
   uNew = smoothing[b, u, 2.0 / 3.0, 1];
   T = SparseArray[{{i_, i_} → 2.0, {i_, j_} /; Abs[i - j] == 1 → -1.0}, {n, n}];
   r = T.uNew - b;
   d = interpolation[MGV[4.0 restriction[r], ConstantArray[0.0, (n - 1) / 2]]];
   uNew = uNew - d;
   uNew = smoothing[b, uNew, 2.0 / 3.0, 2];
  ];
  Return[uNew];
 ]
```

To test one V-cycle we use as an initial guess a straight line satisfying the boundary conditions:

In[26]:=
```
u0[x_] = InterpolatingPolynomial[{{a, ua}, {b, ub}}, x]
x0 = N[u0[Xin]];
```

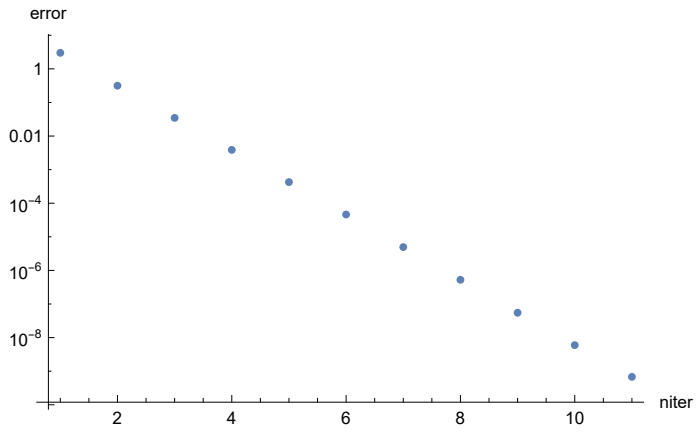
Out[26]=
```
1 + 2 x
```

Let us apply the V-cycle recursively several times and see how the error is decreased. We compare the iterative solution with the direct one obtained above:

In[28]:=
```
niter = 10;
xMGV = x0;
errorMGV = ConstantArray[1.0 × 10^-20, niter + 1];
errorMGV[[1]] = Max[Abs[xMGV - xDirect]];
Do[
 xMGV = MGV[rhs, xMGV];
 errorMGV[[i + 1]] = Max[Abs[xMGV - xDirect]],
 {i, 1, niter}
]
eMGV = xMGV - xDirect;
```

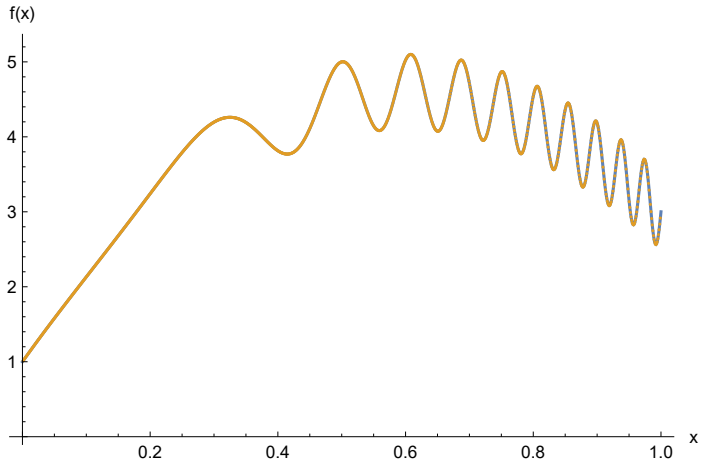
In[77]:=
```
ListLogPlot[{errorMGV}, PlotRange → All, AxesLabel → {"niter", "error"}]
nit1 = 6; nit2 = 10;
ρMGV = Exp[(Log[errorMGV[[nit2]]] - Log[errorMGV[[nit1]]]) / (nit2 - nit1)];
Print["estimated ρ(R_MGV) = ", ρMGV]
ListPlot[{exactSol, Transpose[{Xin, xMGV}]},
 Joined → {True, False}, AxesLabel → {"x", "f(x)"}]
ListPlot[{Transpose[{Xin, eMGV}]}, Joined → {False}, AxesLabel → {"x", "error"}]
```

Out[77]=

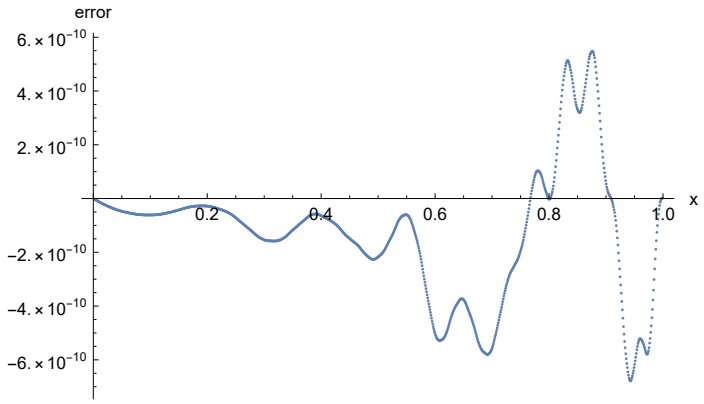



```
estimated ρ(R_MGV) = 0.106649393219
```

Out[81]=



Out[82]=



Full multigrid solution

The full multigrid starts at the level (1) by solving the problem exactly, then we interpolate the result to the next level and run one V-cycle algorithm (MGV), then we again interpolate to the next level and run another V-cycle etc.

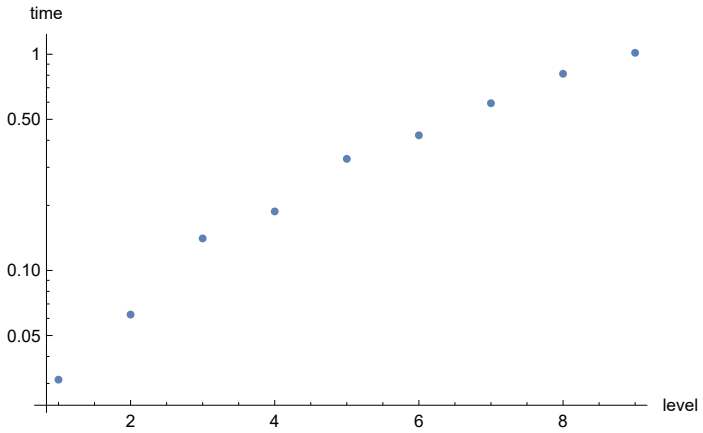
To accomplish this we need a slightly different function for interpolation:

In[52]:=
```
interpolationWithBoundaries[u_, ua_, ub_] :=
 Module[{n, P},
  n = Length[u];
  P = SparseArray[{{i_, j_} /; 2 j - 1 == i → 0.5,
     {i_, j_} /; 2 j - 2 == i → 1.0, {i_, j_} /; 2 j - 3 == i → 0.5}, {2 n + 1, n + 2}];
  Return[P.Join[{ua}, u, {ub}]]
 ]
FMG[k_, fx_, a_, b_, ua_, ub_] :=
 Module[{n, maxk, i2, h, rhs, u},
  n = Length[fx]; maxk = Log[2, n + 1];
  If[k > maxk, Print["Full multigrid can be run only up to the level (", maxk, ")"];];
  h = (b - a) / 2^maxk;
  u = {(-fx[[(n + 1) / 2]] + ua + ub) / 2}; (* exact solution at the level (1) *)
  Do[
   i2 = 2^i;
   h = (b - a) / i2;
   rhs = -h^2 fx[[Range[(n + 1) / i2, n, (n + 1) / i2]]];
   rhs[[1]] = rhs[[1]] + ua;
   rhs[[i2 - 1]] = rhs[[i2 - 1]] + ub;
   u = MGV[rhs, interpolationWithBoundaries[u, ua, ub]],
   {i, 2, Min[k, maxk]}
  ];
  Return[u];
 ]
```

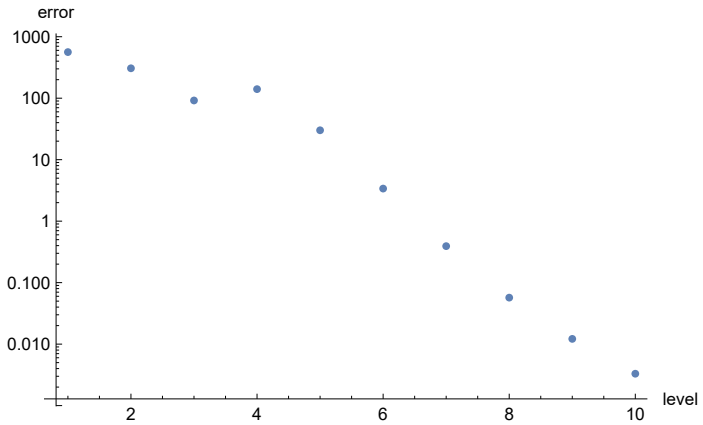
In[54]:=
```
errorFMG = ConstantArray[1.0 × 10^-20, ig];
timeFMG = ConstantArray[0, ig];
Do[
  t = Timing[xFMG = FMG[i, fx, a, b, ua, ub];];
  timeFMG[[i]] = t[[1]];
  h = (b - a) / 2^i;
  Xin = Range[a + h, b - h, h];
  errorFMG[[i]] = Max[Abs[u[Xin] - xFMG]],
  {i, 1, ig}
 ];
eFMG = xFMG - xDirect;
```

In[83]:=
```
ListLogPlot[{timeFMG[[2 ;; ig]]}, PlotRange → All, AxesLabel → {"level", "time"}]
ListLogPlot[{errorFMG}, PlotRange → All, AxesLabel → {"level", "error"}]
ListPlot[{exactSol, Transpose[{Xin, xFMG}]},
 Joined → {True, False}, AxesLabel → {"x", "f(x)"}]
ListPlot[{Transpose[{Xin, eFMG}]}, Joined → {False}, AxesLabel → {"x", "error"}]
```

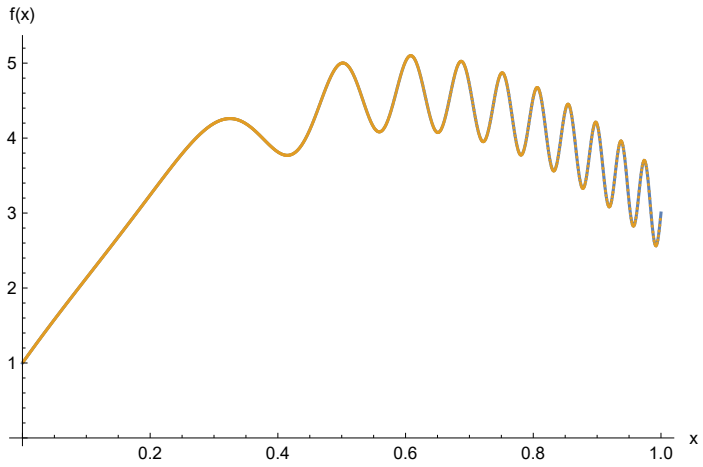
Out[83]=



Out[84]=



Out[85]=



Out[86]=