CP315 Portfolio

Jasmaan Panesar (140722880)

Winter 2018

Contents

7 Interpolation, Curve Fitting

7.1 Vandermonde-based collocation

Consider the function $f(x) = \dfrac{30}{1+3x^2}$, defined over the interval $[-3,+3]$. Using the 7 data points

$$(-3, f(-3)), (-2, f(-2)), (-1, f(-1)), (0, f(0)), (1, f(1)), (2, f(2)), (3, f(3)),$$

compute the corresponding interpolation polynomial of degree 6, using Vandermonde-based collocation.

7.2 Lagrange interpolation

Consider the function $f(x) = \dfrac{30}{1+3x^2}$, defined over the interval $[-3,+3]$. Using the 7 data points

$$(-3, f(-3)), (-2, f(-2)), (-1, f(-1)), (0, f(0)), (1, f(1)), (2, f(2)), (3, f(3)),$$

compute the corresponding interpolation polynomial of degree 6, using Lagrange interpolation.

7.3 Comparison of the function and the interpolation polynomial(s)

Show that the two interpolation polynomials obtained in subsections 7.1 and 7.2 coincide.

Plot the function $f(x)$ and its interpolation polynomial over the interval $[-3,+3]$.

7.4 2D Lagrange interpolation

Consider a function $f(x,y)$ whose values are known at all the $(m+1)(n+1)$ pairs formed by $x_0, x_1, \ldots, x_m$ and $y_0, y_1, \ldots, y_n$. Then this function can be approximated by the 2D Lagrange interpolation polynomial

$$L(x,y) = \sum_{i=0}^{m}\sum_{j=0}^{n} X_i(x)Y_j(y)f(x_i,y_j)$$

where

$$X_i(x) = \prod_{k=0,k\neq i}^{m} \frac{x-x_k}{x_i-x_k}, \quad i=0,1,\ldots,m \quad \& \quad Y_j(y) = \prod_{k=0,k\neq j}^{n} \frac{y-y_k}{y_j-y_k}, \quad j=0,1,\ldots,n$$

are the Lagrange interpolation coefficients.
Find the 2D Lagrange interpolation polynomial $L(x,y)$ corresponding to the following data:

y_j, x_i	0	1	2
0	1	2	7
2	7	20	54
4	54	148	403

Use the polynomial that you found to estimate the function value $f(1.5,3)$.

```
> with(LinearAlgebra) :

> vandermonde := proc(points)
      local p, dpamount, M, y, i, j, row, a, f;
      p := points;
      dpamount := nops(p);
      M := [ ];
      y := [ ];
      for i to dpamount do
          row := [ ];
          for j from 0 to dpamount - 1 do row := [op(row), points[i, 1]^j] end do;
          M := [op(M), row]
      end do;
      M := Matrix(M);
      for i to nops(points) do
          y := [op(y), points[i, 2]]
      end do;
      y := convert(y, Vector);
      a := LinearAlgebra:-LinearSolve(M, y);
      f := 0;
      for i to numelems(a) do
          f := f + a[i]*x^(i - 1)
      end do;
      return f
  end proc
```

```
vandermonde := proc(points)                                              (1)
    local p, dpamount, M, y, i, j, row, a, f;
    p := points;
    dpamount := nops(p);
    M := [ ];
    y := [ ];
    for i to dpamount do
        row := [ ];
        for j from 0 to dpamount - 1 do row := [op(row), points[i, 1]^j] end do;
        M := [op(M), row]
    end do;
    M := Matrix(M);
    for i to nops(points) do y := [op(y), points[i, 2]] end do;
    y := convert(y, Vector);
    a := LinearAlgebra:-LinearSolve(M, y);
    f := 0;
    for i to numelems(a) do f := f + a[i]*x^(i - 1) end do;
    return f
end proc
```

```
> f := 30/(1+3*x^2);
```

$$f := \frac{30}{3x^2+1} \tag{2}$$

```
> points := [[-3, eval(f, x=-3)], [-2, eval(f, x=-2)], [-1, eval(f, x=-1)], [0, eval(f, x=0)],
      [1, eval(f, x=1)], [2, eval(f, x=2)], [3, eval(f, x=3)]];
```

$$points := \left[\left[-3, \frac{15}{14}\right], \left[-2, \frac{30}{13}\right], \left[-1, \frac{15}{2}\right], [0, 30], \left[1, \frac{15}{2}\right], \left[2, \frac{30}{13}\right], \left[3, \frac{15}{14}\right]\right] \tag{3}$$

```
> vandermonde(points);
```

$$30 - \frac{5445}{182}x^2 + \frac{5805}{728}x^4 - \frac{405}{728}x^6 \tag{4}$$

```
> lagrange :=proc(points)
      local equation, num, denom, dpamount, xvalue, yvalue, i, j, c;
      equation := 0;
      num := 1;
      denom := 1;
      dpamount := nops(points);
      for i to dpamount do
          yvalue := points[i][2];
          xvalue := points[i][1];
          for j to dpamount do
              if j<>i then
                  num := num*(x - points[j][1]); denom := denom*(xvalue - points[j][1])
              end if
          end do;
          c := yvalue*num/denom;
          equation := equation + c;
          num := 1;
          denom := 1
      end do;
      return simplify(equation)
  end proc
```

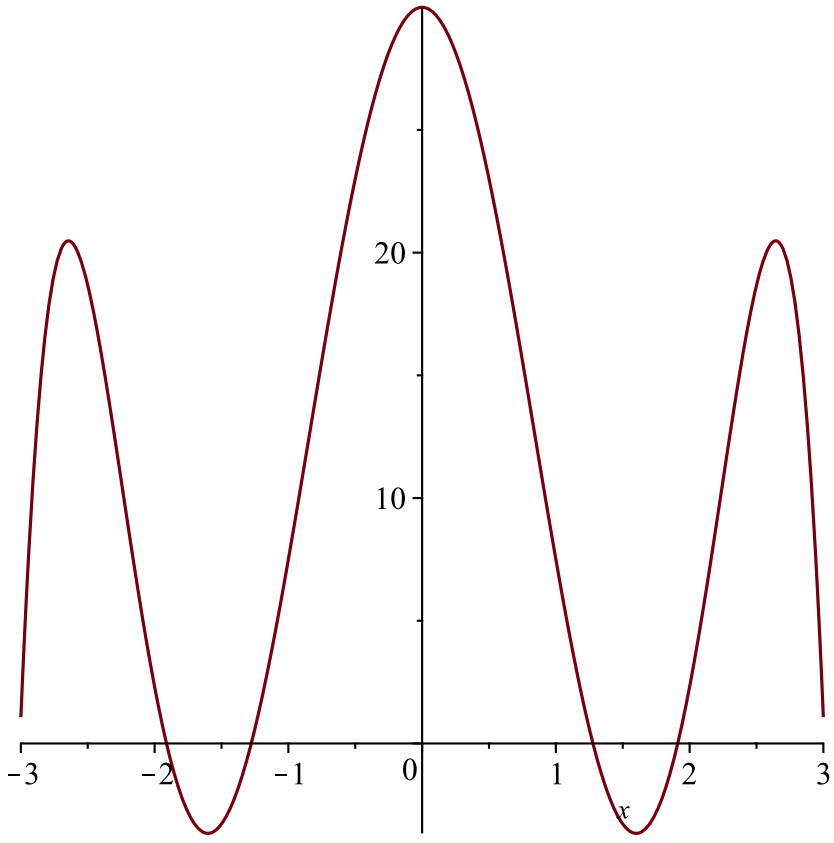
```
lagrange := proc(points)                                                    (5)
    local equation, num, denom, dpamount, xvalue, yvalue, i, j, c;
    equation := 0;
    num := 1;
    denom := 1;
    dpamount := nops(points);
    for i to dpamount do
        yvalue := points[i][2];
        xvalue := points[i][1];
        for j to dpamount do
            if j<>i then
                num := num*(x - points[j][1]); denom := denom*(xvalue - points[j][1])
            end if
```

```
        end do;
        c := yvalue * num / denom;
        equation := equation + c;
        num := 1;
        denom := 1
    end do;
    return simplify(equation)
end proc
```

```
> interpolationPolynomial := lagrange(points);
```

$$interpolationPolynomial := 30 - \frac{5445}{182}\,x^2 + \frac{5805}{728}\,x^4 - \frac{405}{728}\,x^6 \qquad \textbf{(6)}$$

```
> plot(interpolationPolynomial, x = -3 ..3);
```




```
> multiVariateLagrange := proc(points, m, n)
  local i, j, k, xi, yj, f;
  local Xi := 1, Yj := 1;
  local equation := 0;
  for i from 1 to m do:
  Xi := 1;
  xi := points[i·n][1];
```

```
    for k from 1 to m do:
      if k ≠ i then:
        Xi := Xi·( (x − points[k·n][1]) / (xi − points[k·n][1]) );
      end if;
    end do:

    for j from 1 to n do:
      Yj := 1;
      yj := points[j][2];
        for k from 1 to n do:
      if k ≠ j then:
        Yj := Yj·( (y − points[k][2]) / (yj − points[k][2]) );
      end if;
    end do:
    f := points[((i − 1)·3) + j][3];
    equation := equation + f·Xi·Yj;

    end do:

  end do:
  return simplify(equation);
  end proc;
```

```
multiVariateLagrange := proc(points, m, n)                                    (7)
  local i, j, k, xi, yj, f, Xi, Yj, equation;
  Xi := 1;
  Yj := 1;
  equation := 0;
  for i to m do
      Xi := 1;
      xi := points[i * n][1];
      for k to m do
          if k <> i then Xi := Xi * (x − points[k * n][1]) / (xi − points[k * n][1]) end if
      end do;
      for j to n do
          Yj := 1;
          yj := points[j][2];
          for k to n do
              if k <> j then Yj := Yj * (y − points[k][2]) / (yj − points[k][2]) end if
          end do;
          f := points[3 * i − 3 + j][3];
          equation := equation + f * Xi * Yj
      end do
  end do;
  return simplify(equation)
```

```
end proc
> points := [[0, 0, 1], [0, 2, 7], [0, 4, 54], [1, 0, 2], [1, 2, 20], [1, 4, 148], [2, 0, 7], [2, 2, 54],
      [2, 4, 403]];
```

$$points := [[0, 0, 1], [0, 2, 7], [0, 4, 54], [1, 0, 2], [1, 2, 20], [1, 4, 148], [2, 0, 7], [2, 2, 54], [2, 4, 403]] \quad \textbf{(8)}$$

```
> polynomial := multiVariateLagrange(points, 3, 3);
```

$$polynomial := \frac{(123\,y^2 - 178\,y + 32)\,x^2}{16} + \frac{(15\,y^2 - 2\,y - 16)\,x}{16} + \frac{41\,y^2}{8} - \frac{29\,y}{4} + 1 \quad \textbf{(9)}$$

```
> eval(polynomial, [x = 1.5, y = 3]);
```

$$121.0468750 \quad \textbf{(10)}$$

```
>
```

8 Non-Linear Regression

Study carefully the Maple code provided in the next 8 pages, that applies non-linear regression to approximate the trigonometric function

$$\cos\left(\frac{\pi x}{50}\right)\sin\left(\frac{\pi x}{50}\right)$$

in the interval $[0, 4]$.

Modify the code to apply non-linear regression to approximate an arbitrary trigonometric function made of cos and sin terms. Your code should be able to work with an arbitrary number of higher frequency terms.

Test your code with the following trigonometric function:

$$\cos\left(x^2\pi\right) + \cos\left(2\,x^2\right) + \sin\left(x^2\pi\right) + \sin\left(2\,x^2\right)$$

in the interval $[0, 4]$.

How many higher frequency terms do you need to add into your model, in order to get a reasonable approximation of this function? Illustrate the quality of your approximation by a plot similar to the ones contained in the Maple code, i.e. overlapping the sampling data and the data produced by the model.

```
> restart;
  n := 50;
  f := cos((i^2)*Pi) + cos((i^2)*2) + sin((i^2)*Pi) + sin((i^2)*2);
  omega := Pi/10;
   terms := 85;
```

$$n := 50$$

$$f := \cos(i^2\,\pi) + \cos(2\,i^2) + \sin(i^2\,\pi) + \sin(2\,i^2)$$

$$\omega := \frac{\pi}{10}$$

$$terms := 85 \tag{1}$$

```
> x := [seq(Pi*i/n, i=1..n)];
```

$$x := \left[\frac{\pi}{50}, \frac{\pi}{25}, \frac{3\,\pi}{50}, \frac{2\,\pi}{25}, \frac{\pi}{10}, \frac{3\,\pi}{25}, \frac{7\,\pi}{50}, \frac{4\,\pi}{25}, \frac{9\,\pi}{50}, \frac{\pi}{5}, \frac{11\,\pi}{50}, \frac{6\,\pi}{25}, \frac{13\,\pi}{50}, \frac{7\,\pi}{25}, \frac{3\,\pi}{10}, \right. \tag{2}$$

$$\frac{8\,\pi}{25}, \frac{17\,\pi}{50}, \frac{9\,\pi}{25}, \frac{19\,\pi}{50}, \frac{2\,\pi}{5}, \frac{21\,\pi}{50}, \frac{11\,\pi}{25}, \frac{23\,\pi}{50}, \frac{12\,\pi}{25}, \frac{\pi}{2}, \frac{13\,\pi}{25}, \frac{27\,\pi}{50}, \frac{14\,\pi}{25},$$

$$\frac{29\,\pi}{50}, \frac{3\,\pi}{5}, \frac{31\,\pi}{50}, \frac{16\,\pi}{25}, \frac{33\,\pi}{50}, \frac{17\,\pi}{25}, \frac{7\,\pi}{10}, \frac{18\,\pi}{25}, \frac{37\,\pi}{50}, \frac{19\,\pi}{25}, \frac{39\,\pi}{50}, \frac{4\,\pi}{5}, \frac{41\,\pi}{50},$$

$$\left.\frac{21\,\pi}{25}, \frac{43\,\pi}{50}, \frac{22\,\pi}{25}, \frac{9\,\pi}{10}, \frac{23\,\pi}{25}, \frac{47\,\pi}{50}, \frac{24\,\pi}{25}, \frac{49\,\pi}{50}, \pi\right]$$

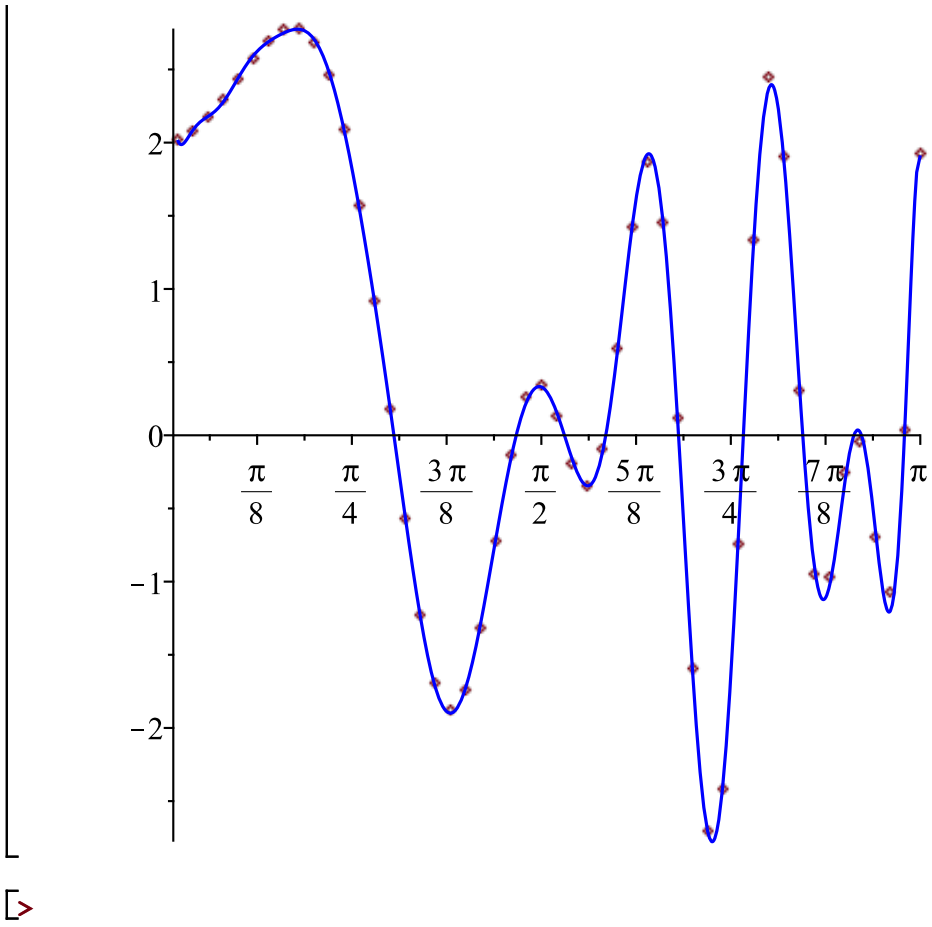
```
> y := [seq(evalf(eval(f, i=x[j])), j=1..n)];
```

$$y := [2.020189713, 2.079438161, 2.173645188, 2.295541658, 2.434126919, 2.574076967, \tag{3}$$
$$2.695346213, 2.773272660, 2.779571193, 2.684625599, 2.461411918, 2.091137956,$$
$$1.570210122, 0.9174300218, 0.1794706468, -0.5680688404, -1.226956137,$$
$$-1.692049614, -1.876153197, -1.739732878, -1.316476202, -0.7221118364,$$
$$-0.1344475681, 0.2599709526, 0.3422100292, 0.1313796720, -0.1923305610,$$
$$-0.3472117741, -0.0935597782, 0.5929920219, 1.422184519, 1.868711943,$$
$$1.455292944, 0.1181311518, -1.592372178, -2.705322794, -2.418660639,$$
$$-0.7422744198, 1.334708050, 2.447868586, 1.904179156, 0.3051301709,$$
$$-0.9476191242, -0.9691867601, -0.2557676954, -0.0420613979, -0.6950614666,$$
$$-1.071729286, 0.0375174331, 1.925507114]$$

```
> L := [seq(a[i], i=1..terms)]:
  sExpression := 0:
   m := 1:
   for i from 1 to terms do:
      if (i mod 2 = 1) then
         sExpression := sExpression + L[i]·sin(m·omega·t);
      else:
         sExpression := sExpression + L[i]·cos(m·omega·t);
         m := m + 1;
      end if
   end do:
   i :='i':
   S := evalf(sum((y[i]-eval(sExpression, t=x[i]))^2, i=1..n)):
    i :='i':
```

```
SL := [ ]:
for i from 1 to terms do:
  SL := [op(SL), diff(S, L[i])]:
end do:
sols := solve({seq(SL[i], i = 1..terms)}, {seq(L[i], i = 1..terms)});
assign(sols):
i :='i':
dataPoints := plot([seq([x[i], y[i]], i = 1..n)], style = point):
lb := min(op(x)):
ub := max(op(x)):
quadraticRegression := plot(sExpression, t = lb..ub, color = blue):
plots[display]({dataPoints, quadraticRegression});
```

$$
\begin{aligned}
sols := \{&a_1 = -1038.513525, a_2 = 966.0239791, a_3 = -452.9143123, a_4 = 375.4461572, a_5 \\
&= 425.4973454, a_6 = -996.9456620, a_7 = -799.3329744, a_8 = 318.3204132, a_9 \\
&= -73.00261676, a_{10} = 963.4460711, a_{11} = 198.6108278, a_{12} = -913.6216556, a_{13} \\
&= -672.7067029, a_{14} = -934.8635773, a_{15} = 118.4870516, a_{16} = -242.5174147, a_{17} \\
&= 99.33342112, a_{18} = 1729.980138, a_{19} = 2332.580171, a_{20} = -1851.003479, a_{21} \\
&= -1344.523835, a_{22} = 1724.977014, a_{23} = -1519.872093, a_{24} = 390.7505990, a_{25} \\
&= 394.3280197, a_{26} = -684.3007360, a_{27} = 339.1182558, a_{28} = 767.1336960, a_{29} \\
&= -762.0902873, a_{30} = -1301.913318, a_{31} = -250.9578638, a_{32} = 190.2034633, a_{33} \\
&= 163.9914526, a_{34} = -224.7237463, a_{35} = 582.9939467, a_{36} = 228.5992049, a_{37} \\
&= -772.4679940, a_{38} = 30.45820655, a_{39} = 316.2503366, a_{40} = -860.7141956, a_{41} \\
&= -395.0775057, a_{42} = 274.3434980, a_{43} = 626.3182886, a_{44} = 757.5583568, a_{45} \\
&= -943.4616648, a_{46} = -1285.321775, a_{47} = 102.5608263, a_{48} = -200.2765399, a_{49} \\
&= -86.46706587, a_{50} = -258.7661860, a_{51} = 751.2270966, a_{52} = -1107.167763, a_{53} \\
&= 1151.697382, a_{54} = -350.4773248, a_{55} = 2121.748654, a_{56} = 562.8095993, a_{57} \\
&= -1382.791137, a_{58} = 2546.691291, a_{59} = 300.5936662, a_{60} = -2.811667238, a_{61} \\
&= -798.6607960, a_{62} = -793.9090886, a_{63} = 197.8373212, a_{64} = 1816.498971, a_{65} \\
&= -1013.720355, a_{66} = -438.6857161, a_{67} = -902.4973633, a_{68} = -624.0483181, a_{69} \\
&= 285.1749789, a_{70} = -403.3600711, a_{71} = 278.2341512, a_{72} = -1422.525026, a_{73} \\
&= 990.1807526, a_{74} = 1907.360254, a_{75} = -425.6978104, a_{76} = -1024.088324, a_{77} \\
&= -475.9551810, a_{78} = 427.3952531, a_{79} = 670.8979064, a_{80} = 299.2334712, a_{81} \\
&= -683.9951294, a_{82} = -594.0578365, a_{83} = 479.2376696, a_{84} = 242.9958581, a_{85} \\
&= -123.4106422\}
\end{aligned}
$$


2
1
0
−1
−2
π/8
π/4
3π/8
π/2
5π/8
3π/4
7π/8
π

9 Numerical Integration I, Newton-Cotes

Write a Maple or Matlab routine to implement the multi-step **Trapezoidal Rule**.

Write a Maple or Matlab routine to implement the multi-step **Simpson's 1/3 Rule**.

Both your routines should accept as input, the function $f(x)$ to be integrated, the limits of integration a, b, the accuracy ϵ and the number of n subintervals specified by the $n+1$ data points: $a = x_0, x_1, \ldots, x_{n-1}, x_n = b$.

Implement the following **optimization strategy**: if the routine is called with **double** the number of subintervals, i.e. $4, 8, 16$, etc, then the routine should not recompute the function evaluations of points that have been computed already.

Implement the following **convergence concept**: if two successive doublings result in two numerical values which are less than ϵ apart, then the algorithm has converged and the latest value computed should be returned as the approximate value of the integral.

Use your two routines to compute numerically the following definite integrals, starting with $n = 2$.

$I_1 = \int_0^1 \frac{x^2+x+1}{x^4+x^3+x^2+x+1} dx$	0.86480626597720996728
$I_2 = \int_0^{\frac{\pi}{2}} \frac{\sin^3(\theta)}{\sin^3(\theta)+\cos^3(\theta)} d\theta$	$\frac{\pi}{4}$
$I_3 = \int_{-\pi}^{\pi} \cos(x^2) dx$	1.1313870272133648866

Compare your results with the exact values, given in the second column above.

Show the results of your work in a tabular format as follows:

integral	# doublings to achieve conv. $\epsilon = 10^{-5}$	# doublings to achieve conv. $\epsilon = 10^{-8}$
I_1 Trapez.	...	...
I_2 Trapez.	...	...
I_3 Trapez.	...	...
I_1 Simpson	...	...
I_2 Simpson	...	...
I_3 Simpson	...	...

Answer		
integral	doublings to achieve conv. E = $10^{-}5$	doublings to achieve conv. E = $10^{-}8$
$I_1 Trapez.$	7	12
$I_2 Trapez.$	2	2
$I_3 Trapez.$	12	17
$I_1 Simpson.$	4	7
$I_2 Simpson.$	2	2
$I_3 Simpson.$	8	11

```
> trap := proc(a, b, n, f, e)
      local h, xVal, expression, value, lastvalue, nVal, doublings, doubleh;
      lastvalue := 0;
      value := 1 + e;
      nVal := n;
      doublings := 0;
      expression := eval(f, x = a) + eval(f, x = b);
      doubleh := 1;
      while e < abs(evalf(value) - evalf(lastvalue)) do
          h := (b - a) / nVal;
          xVal := a + h;
          while evalf(xVal) < evalf(b) do
              expression := expression + 2 * (eval(f, x = xVal)); xVal := xVal + doubleh * h
          end do;
          lastvalue := value;
          value := 1 / 2 * h * expression;
          nVal := 2 * nVal;
          doublings := doublings + 1;
          doubleh := 2
      end do;
      return value, doublings
  end proc:
```

```
> I1 := (x^2 + x + 1) / (x^4 + x^3 + x^2 + x + 1); a1 := 0; b1 := 1;
```

$$I_1 := \frac{x^2 + x + 1}{x^4 + x^3 + x^2 + x + 1}$$

$$a_1 := 0$$

$$b_1 := 1 \qquad (1)$$

```
> I2 := sin(x)^3 / (sin(x)^3 + cos(x)^3); a2 := 0; b2 := Pi / 2;
```

$$I_2 := \frac{\sin(x)^3}{\sin(x)^3 + \cos(x)^3}$$

$$a_2 := 0$$

$$b_2 := \frac{\pi}{2} \qquad (2)$$

```
> I3 := cos(x^2); a3 := -Pi; b3 := Pi;
```

$$I_3 := \cos(x^2)$$

$$a_3 := -\pi$$

$$b_3 := \pi \qquad (3)$$

```
> evalf(trap(a1, b1, 2, I1, 10^-5));
```

$$0.8648032142, 7. \qquad (4)$$

```
> evalf(trap(a2, b2, 2, I2, 10^-5)); evalf(Pi/4);
```

0.7853981638, 2.

0.7853981635 **(5)**

```
> evalf(trap(a3, b3, 2, I3, 10^-5));
```

1.131388085, 12. **(6)**

```
> evalf(trap(a1, b1, 2, I1, 10^-8));
```

0.8648062630, 12. **(7)**

```
> evalf(trap(a2, b2, 2, I2, 10^-8)); evalf(Pi/4);
```

0.7853981638, 2.

0.7853981635 **(8)**

```
> evalf(trap(a3, b3, 2, I3, 10^-8));
```

1.131387036, 17. **(9)**

```
>
```

```
> simpson := proc(a, b, n, f, e)
      local h, xVal, expression, value, lastvalue, nVal, doublings, doubleh, expressionNew;
      lastvalue := 0;
      value := 1 + e;
      nVal := n;
      doublings := 0;
      expression := 0;
      doubleh := 1;
      while e < abs(evalf(value) - evalf(lastvalue)) do
          h := (b - a)/nVal;
          xVal := a + h;
          expressionNew := 0;
          while evalf(xVal) < evalf(b) do
              expressionNew := expressionNew + eval(f, x = xVal); xVal := xVal + doubleh * h
          end do;
          lastvalue := value;
          value := 1/3 * h * (4 * expressionNew + 2 * expression + eval(f, x = a) + eval(f, x
          = b));
          expression := expression + expressionNew;
          nVal := 2 * nVal;
          doublings := doublings + 1;
          doubleh := 2
      end do;
      return value, doublings
  end proc
```

```
simpson := proc(a, b, n, f, e)                                                    (1)
    local h, xVal, expression, value, lastvalue, nVal, doublings, doubleh, expressionNew;
    lastvalue := 0;
    value := 1 + e;
    nVal := n;
    doublings := 0;
    expression := 0;
    doubleh := 1;
    while e < abs(evalf(value) - evalf(lastvalue)) do
        h := (b - a)/nVal;
        xVal := a + h;
        expressionNew := 0;
        while evalf(xVal) < evalf(b) do
            expressionNew := expressionNew + eval(f, x = xVal); xVal := xVal + doubleh * h
        end do;
        lastvalue := value;
        value := 1/3 * h * (4 * expressionNew + 2 * expression + eval(f, x = a) + eval(f, x
        = b));
        expression := expression + expressionNew;
        nVal := 2 * nVal;
        doublings := doublings + 1;
```

```
        doubleh := 2
    end do;
    return value, doublings
end proc
```

> $I_1 := \frac{(x^2+x+1)}{x^4+x^3+x^2+x+1}; a_1 := 0; b_1 := 1;$

$$I_1 := \frac{x^2+x+1}{x^4+x^3+x^2+x+1}$$
$$a_1 := 0$$
$$b_1 := 1 \quad (2)$$

> $I_2 := \frac{\sin^3(x)}{\sin^3(x)+\cos^3(x)}; a_2 := 0; b_2 := \frac{\mathrm{Pi}}{2};$

$$I_2 := \frac{\sin(x)^3}{\sin(x)^3+\cos(x)^3}$$
$$a_2 := 0$$
$$b_2 := \frac{\pi}{2} \quad (3)$$

> $I_3 := \cos(x^2); a_3 := -\mathrm{Pi}; b_3 := \mathrm{Pi};$

$$I_3 := \cos(x^2)$$
$$a_3 := -\pi$$
$$b_3 := \pi \quad (4)$$

> $evalf(simpson(a_1, b_1, 2, I_1, 10^{-5}));$

$$0.8648068803, 4. \quad (5)$$

> $evalf(simpson(a_2, b_2, 2, I_2, 10^{-5})); evalf\left(\frac{\mathrm{Pi}}{4}\right);$

$$0.7853981634, 2.$$
$$0.7853981635 \quad (6)$$

> $evalf(simpson(a_3, b_3, 2, I_3, 10^{-5}));$

$$1.131386735, 8. \quad (7)$$

> $evalf(simpson(a_1, b_1, 2, I_1, 10^{-8}));$

$$0.8648062661, 7. \quad (8)$$

> $evalf(simpson(a_2, b_2, 2, I_2, 10^{-8})); evalf\left(\frac{\mathrm{Pi}}{4}\right);$

$$0.7853981634, 2.$$
$$0.7853981635 \quad (9)$$

> $evalf(simpson(a_3, b_3, 2, I_3, 10^{-8}));$

$$1.131387027, 11. \quad (10)$$

10 Numerical Integration II, Romberg

Romberg Integration is a numerical integration method based on forming a triangular array of approximations to a definite integral $I = \int_a^b f(x)\,dx$. This triangular array is formed by computing trapezoidal approximations with a variable spacing for I and subsequently computing their Richardson extrapolates.

10.1 Description of the method

Denote by $T_k^{(0)}$ the trapezoidal approximation to I with spacing $h_k = (b-a)/2^k$, i.e.

$$T_k^{(0)} = \frac{h_k}{2}[(f_0 + f_n) + 2(f_1 + \cdots + f_{n-1})]$$

where $n = 2^k$ and $f_m = f(a + mh)$, $m = 0, \ldots, n$.
Denote by $T_k^{(1)}$ the Richardson extrapolate of $T_k^{(0)}$ and $T_{k+1}^{(0)}$, i.e. $T_k^{(1)} = \frac{2^2 T_{k+1}^{(0)} - T_k^{(0)}}{2^2 - 1}$
Denote by $T_k^{(2)}$ the Richardson extrapolate of $T_k^{(1)}$ and $T_{k+1}^{(1)}$, i.e. $T_k^{(2)} = \frac{2^4 T_{k+1}^{(1)} - T_k^{(1)}}{2^4 - 1}$
In general, define $T_k^{(m)} = \frac{4^m T_{k+1}^{(m-1)} - T_k^{(m-1)}}{4^m - 1}$ for $m = 1, 2, \ldots$.
Using $T_k^{(m)}$ recursively, we can construct the following triangular array of approximations to the integral I :

$$\begin{array}{cccc} T_0^{(0)} & & & \\ T_1^{(0)} & T_0^{(1)} & & \\ T_2^{(0)} & T_1^{(1)} & T_0^{(2)} & \\ \cdots & \cdots & \cdots & \cdots \end{array}$$

Romberg integration stipulates that the sequences of elements in all columns of the above array (as well as the sequence of diagonal elements $T_0^{(0)}, T_0^{(1)}, T_0^{(2)}, \ldots$) converge to the same limit, which is the value of I. Write a Maple or Matlab routine to implement Romberg integration.

10.2 Questions

1. Compute the value of the definite integral $I = \int_0^1 \frac{1}{1+x}\,dx$ correct up to 10 decimal digits, using Romberg integration. Use the exact result $I = \ln 2$ to check your answer. How many rows of the Romberg array did you need to compute, to obtain 10 decimal digits of accuracy? Show clearly all the elements of the triangular Romberg array that you computed.

2. Compute the value of the definite integral $I = \int_0^{\frac{\pi}{2}} \frac{\sin^3(\theta)}{\sin^3(\theta) + \cos^3(\theta)}\,d\theta$ correct up to 10 decimal digits, using Romberg integration. Use the exact result $I = \frac{\pi}{4}$ to check your answer. How many rows of the Romberg array did you need to compute, to obtain 10 decimal digits of accuracy? Show clearly all the elements of the triangular Romberg array that you computed.

Answer:
Please see following pages for Maple code corresponding to the results below:

- $I = \int_0^1 \frac{1}{1+x} = ln(2) = 0.6931471806$, requires 7 rows of Romberg Array to be computed.
- $I = \int_0^{\pi/2} \frac{\sin\theta^3}{\sin\theta^3+\cos\theta^3} = \pi/4 = 0.7853981635$, requires 2 rows of Romberg Array to be computed.

```
> T := proc(a, b, n, f)
    local h, xVal, expression;    h := (b - a)/n; expression := eval(f, x = a) + eval(f, x = b);
      xVal := a + h;
      while evalf(xVal) < evalf(b) do
          expression := expression + 2 * (eval(f, x = xVal));
          xVal := xVal + h
      end do;
      return h/2 · expression;
    end proc
```

```
T := proc(a, b, n, f)                                                    (1)
  local h, xVal, expression;
  h := (b - a)/n;
  expression := eval(f, x = a) + eval(f, x = b);
  xVal := a + h;
  while evalf(xVal) < evalf(b) do
      expression := expression + 2 * (eval(f, x = xVal)); xVal := xVal + h
  end do;
  return 1/2 * h * expression
end proc
```

```
> Romberg := proc(a, b, f, e)
    local rombergArray := [ ];
    local k := 0;
    local m := 0;
    local row := 1;
    local answerPrevRow := 0;
    local tRow := [ ];
    local answer := T(a, b, 2^k, f);
    tRow := [op(tRow), answer];
    rombergArray := [op(rombergArray), tRow];


    while(abs(evalf(answerPrevRow) - evalf(answer)) > e) do:
     k := k + 1;
     answerPrevRow := answer;
      answer := T(a, b, 2^k, f);
     tRow := [ ];
     tRow := [op(tRow), answer];
     for m from 1 to k do:
       answer := (4^m · tRow[m] - rombergArray[k][m])/(4^m - 1);
       tRow := [op(tRow), answer];
    end do:
   rombergArray := [op(rombergArray), tRow];


    end do:
```

```
    return rombergArray;

    end proc
Romberg := proc(a, b, f, e)                                                    (2)
    local rombergArray, k, m, row, answerPrevRow, tRow, answer;
    rombergArray := [ ];
    k := 0;
    m := 0;
    row := 1;
    answerPrevRow := 0;
    tRow := [ ];
    answer := T(a, b, 2^k, f);
    tRow := [op(tRow), answer];
    rombergArray := [op(rombergArray), tRow];
    while e < abs(evalf(answerPrevRow) - evalf(answer)) do
        k := k + 1;
        answerPrevRow := answer;
        answer := T(a, b, 2^k, f);
        tRow := [ ];
        tRow := [op(tRow), answer];
        for m to k do
            answer := (4^m * tRow[m] - rombergArray[k][m]) / (4^m - 1);
            tRow := [op(tRow), answer]
        end do;
        rombergArray := [op(rombergArray), tRow]
    end do;
    return rombergArray
end proc
>
                                                                               (3)
> answer := evalf(Romberg(0, 1, 1/(1+x), 10^-10)):
    for i from 1 to numelems(answer) do;
    print(answer[i]);
    end do:

                              [0.7500000000]
                    [0.7083333333, 0.6944444444]
          [0.6970238095, 0.6932539683, 0.6931746032]
```

```
[0.6941218504, 0.6931545307, 0.6931479015, 0.6931474776]
[0.6933912022, 0.6931476528, 0.6931471943, 0.6931471831, 0.6931471819]
[0.6932082083, 0.6931472103, 0.6931471808, 0.6931471806, 0.6931471806, 0.6931471806]
[0.6931624389, 0.6931471824, 0.6931471806, 0.6931471806, 0.6931471806, 0.6931471806,
  0.6931471806]                                                                    (4)
> evalf(ln(2))
                                  0.6931471806                                     (5)
> answer := evalf(Romberg(0, Pi/2, sin^3(x)/(sin^3(x)+cos^3(x)), 10^-10)):
  for i from 1 to numelems(answer) do;
  print(answer[i]);
  end do:
                                  [0.7853981635]
                            [0.7853981635, 0.7853981635]                           (6)
> evalf(Pi/4);
                                  0.7853981635                                     (7)
>
```

11 Numerical Solution of ODEs, IVPs

11.1 One Equation

Solve the following IVP using Maple/Matlab code:

$$\begin{cases} y' = -5y + e^{-2x} \\ y(0) = 1.0 \\ x \in [0, 1] \end{cases}$$

using Euler's method, Heun's method and the third order Runge-Kutta method seen in class.

Compute the exact solution of the above IVP.
For each of the 3 methods, use a stepsize $h = 0.02$, use 4 digits of accuracy in your computations and make the corresponding tables, to compute the error at each point x_i of your subdivision.

Compare the errors of the 3 methods using this particular stepsize. What is your conclusion?

For each of the 3 methods, plot (in the interval $[0, 1]$) the exact solution of the IVP against the numerical solution found by your code, to obtain a visual of the quality of your approximation to the actual exact solution.

11.2 Two Equations

Implement Euler's method in Maple/Matlab to solve the following system of equations in $[0, 1]$:

$$\begin{cases} x_1'(t) = -7x_1(t) + 8x_2(t) \\ x_2'(t) = -6x_1(t) + 7x_2(t) \\ x_1(0) = 5 \\ x_2(0) = 0 \end{cases}$$

using stepsize $h = 0.01$ and 4 digits of accuracy. Plot the exact solutions against the numerical solutions found by your code.

Answer:
As seen in the following pages of Maple code, Runge-Kutta's method provided the best approximation of the IVP, Heun's method provided the second best approximation and Euler's method provided the worst approximation. This is easily visible in the following graphs as the approximations Runge-Kutta's method exactly overlap the exact solution, whereas the graph for Heuler's clearly doesn't overlap. See the following pages of Maple code for the calculations of Euler's method with 2 equations and its corresponding graph as well.

```
> Euler := proc(a, b, h, y0, f)
    local i;
    local n := (b - a)/h;
    local t := a;
    local w := y0;
    local yi := w;
    local e := 0;

    local ode := diff(y(x), x) = f;
    local ics := y(0) = y0;
    local exactSolution := rhs(dsolve({ode, ics}));
    local dataPointsY, dataPointsW;
    local xList := [t];
    local wList := [w];
    local yList := [yi];
   Digits := 4 :

    print(Step, t[i], w[i], y[i], e[i]);
    print(0, t, w, yi, e);

    for i from 1 to n + 1 do:
    yi := evalf(eval(exactSolution, x = t + h));
    w := w + h*evalf(eval(f, [x = t, y = w]));
    e := abs(yi - w);
    print(0, t, w, yi, e);

     xList := [op(xList), t];
    wList := [op(wList), w];
    yList := [op(yList), yi];
    t := t + h;

    end do:
    dataPointsY := plot([seq([xList[i], yList[i]], i = 1 ..n)], style = line, color = red) :
     dataPointsW := plot([seq([xList[i], wList[i]], i = 1 ..n)], style = line, color = blue) :
    plots[display]({dataPointsY, dataPointsW});


    end proc
```

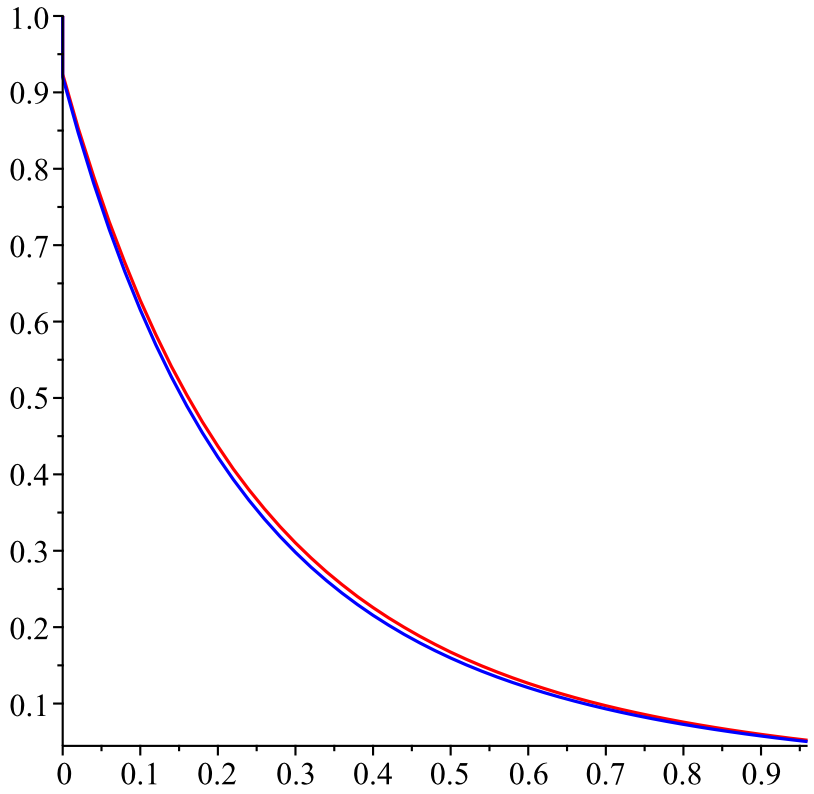
```
Euler := proc(a, b, h, y0, f)                                    (1)
    local i, n, t, w, yi, e, ode, ics, exactSolution, dataPointsY, dataPointsW, xList, wList, yList;
    n := (b - a)/h;
    t := a;
    w := y0;
    yi := w;
    e := 0;
    ode := diff(y(x), x) = f;
    ics := y(0) = y0;
```

```
    exactSolution := rhs(dsolve({ode, ics}));
    xList := [t];
    wList := [w];
    yList := [yi];
    Digits := 4;
    print(Step, t[i], w[i], y[i], e[i]);
    print(0, t, w, yi, e);
    for i to n + 1 do
        yi := evalf(eval(exactSolution, x = t + h));
        w := w + h * evalf(eval(f, [x = t, y = w]));
        e := abs(yi - w);
        print(0, t, w, yi, e);
        xList := [op(xList), t];
        wList := [op(wList), w];
        yList := [op(yList), yi];
        t := t + h
    end do;
    dataPointsY := plot([seq([xList[i], yList[i]], i = 1 ..n)], style = line, color = red);
    dataPointsW := plot([seq([xList[i], wList[i]], i = 1 ..n)], style = line, color = blue);
    plots[display]({dataPointsY, dataPointsW})
end proc
```

```
> Euler(0, 1, 0.02, 1, -5·y(x) + exp(-2·x));
```

$Step, t_i, w_i, y_i, e_i$
0, 0, 1, 1, 0
0, 0, 0.92, 0.9238, 0.0038
0, 0.02, 0.8472, 0.8531, 0.0059
0, 0.04, 0.7809, 0.7897, 0.0088
0, 0.06, 0.7206, 0.7306, 0.0100
0, 0.08, 0.6656, 0.6775, 0.0119
0, 0.10, 0.6154, 0.6278, 0.0124
0, 0.12, 0.5696, 0.5830, 0.0134
0, 0.14, 0.5278, 0.5414, 0.0136
0, 0.16, 0.4895, 0.5038, 0.0143
0, 0.18, 0.4545, 0.4687, 0.0142
0, 0.20, 0.4225, 0.4368, 0.0143
0, 0.22, 0.3931, 0.4069, 0.0138
0, 0.24, 0.3662, 0.3799, 0.0137
0, 0.26, 0.3415, 0.3549, 0.0134
0, 0.28, 0.3188, 0.3317, 0.0129
0, 0.30, 0.2979, 0.3103, 0.0124
0, 0.32, 0.2786, 0.2907, 0.0121
0, 0.34, 0.2609, 0.2724, 0.0115

0, 0.36, 0.2446, 0.2557, 0.0111
0, 0.38, 0.2295, 0.2400, 0.0105
0, 0.40, 0.2155, 0.2256, 0.0101
0, 0.42, 0.2026, 0.2122, 0.0096
0, 0.44, 0.1906, 0.1998, 0.0092
0, 0.46, 0.1795, 0.1882, 0.0087
0, 0.48, 0.1692, 0.1774, 0.0082
0, 0.50, 0.1596, 0.1673, 0.0077
0, 0.52, 0.1507, 0.1580, 0.0073
0, 0.54, 0.1424, 0.1493, 0.0069
0, 0.56, 0.1347, 0.1412, 0.0065
0, 0.58, 0.1275, 0.1336, 0.0061
0, 0.60, 0.1208, 0.1265, 0.0057
0, 0.62, 0.1145, 0.1199, 0.0054
0, 0.64, 0.1086, 0.1136, 0.0050
0, 0.66, 0.1031, 0.1078, 0.0047
0, 0.68, 0.09792, 0.1023, 0.00438
0, 0.70, 0.09306, 0.09718, 0.00412
0, 0.72, 0.08849, 0.09235, 0.00386
0, 0.74, 0.08419, 0.08782, 0.00363
0, 0.76, 0.08014, 0.08353, 0.00339
0, 0.78, 0.07633, 0.07951, 0.00318
0, 0.80, 0.07274, 0.07568, 0.00294
0, 0.82, 0.06935, 0.07215, 0.00280
0, 0.84, 0.06614, 0.06876, 0.00262
0, 0.86, 0.06311, 0.06554, 0.00243
0, 0.88, 0.06024, 0.06252, 0.00228
0, 0.90, 0.05752, 0.05964, 0.00212
0, 0.92, 0.05494, 0.05693, 0.00199
0, 0.94, 0.05250, 0.05435, 0.00185
0, 0.96, 0.05018, 0.05194, 0.00176
0, 0.98, 0.04798, 0.04962, 0.00164
0, 1.00, 0.04589, 0.04742, 0.00153




```
> Heun := proc(a, b, h, y0, f)
  local i;
  local n := (b - a)/h;
  local t := a;
  local w := y0;
  local yi := w;
  local e := 0;
  local wt;
  local ode := diff(y(x), x) = f;
  local ics := y(0) = y0;
  local exactSolution := rhs(dsolve({ode, ics}));

   local dataPointsY, dataPointsW;
  local xList := [t];
  local wList := [w];
   local yList := [yi];

  print(Step, t[i], w[i], y[i], e[i]);
```

```
    print(0, t, w, yi, e);

    for i from 1 to n + 1 do:
    yi := evalf(eval(exactSolution, x = t + h));
    wt := evalf(w + h·eval(f, [x = t, y = w]));
    w := evalf(w + h/2 · (eval(f, [x = t, y = w]) + eval(f, [x = t, y = wt])));
    e := abs(yi − w);
    print(0, t, w, yi, e);

    xList := [op(xList), t];
    wList := [op(wList), w];
    yList := [op(yList), yi];
    t := t + h;


    end do:
    dataPointsY := plot([seq([xList[i], yList[i]], i = 1 ..n)], style = line, color = red):
    dataPointsW := plot([seq([xList[i], wList[i]], i = 1 ..n)], style = line, color = blue):
    plots[display]({dataPointsY, dataPointsW});
    end proc
```

(2)

```
Heun := proc(a, b, h, y0, f)
    local i, n, t, w, yi, e, wt, ode, ics, exactSolution, dataPointsY, dataPointsW, xList, wList, yList;
    n := (b − a)/h;
    t := a;
    w := y0;
    yi := w;
    e := 0;
    ode := diff(y(x), x) = f;
    ics := y(0) = y0;
    exactSolution := rhs(dsolve({ode, ics}));
    xList := [t];
    wList := [w];
    yList := [yi];
    print(Step, t[i], w[i], y[i], e[i]);
    print(0, t, w, yi, e);
    for i to n + 1 do
        yi := evalf(eval(exactSolution, x = t + h));
        wt := evalf(w + h * (eval(f, [x = t, y = w])));
        w := evalf(w + 1/2 * h * (eval(f, [x = t, y = w]) + eval(f, [x = t, y = wt])));
        e := abs(yi − w);
        print(0, t, w, yi, e);
        xList := [op(xList), t];
        wList := [op(wList), w];
        yList := [op(yList), yi];
        t := t + h
```

```
    end do;
    dataPointsY := plot( [seq( [xList[i], yList[i]], i = 1 ..n ) ], style = line, color = red);
    dataPointsW := plot( [seq( [xList[i], wList[i]], i = 1 ..n ) ], style = line, color = blue);
    plots[display]( {dataPointsY, dataPointsW} )
end proc
```

```
> Digits := 4; Heun(0, 1, 0.02, 1, -5·y(x) + exp( -2·x) );
```

$$Digits := 4$$

$$Step,\ t_i,\ w_i,\ y_i,\ e_i$$

0, 0, 1, 1, 0

0, 0, 0.9240, 0.9238, 0.0002

0, 0.02, 0.8545, 0.8531, 0.0014

0, 0.04, 0.7908, 0.7897, 0.0011

0, 0.06, 0.7325, 0.7306, 0.0019

0, 0.08, 0.6791, 0.6775, 0.0016

0, 0.10, 0.6301, 0.6278, 0.0023

0, 0.12, 0.5852, 0.5830, 0.0022

0, 0.14, 0.5440, 0.5414, 0.0026

0, 0.16, 0.5061, 0.5038, 0.0023

0, 0.18, 0.4713, 0.4687, 0.0026

0, 0.20, 0.4393, 0.4368, 0.0025

0, 0.22, 0.4098, 0.4069, 0.0029

0, 0.24, 0.3826, 0.3799, 0.0027

0, 0.26, 0.3576, 0.3549, 0.0027

0, 0.28, 0.3345, 0.3317, 0.0028

0, 0.30, 0.3132, 0.3103, 0.0029

0, 0.32, 0.2935, 0.2907, 0.0028

0, 0.34, 0.2752, 0.2724, 0.0028

0, 0.36, 0.2583, 0.2557, 0.0026

0, 0.38, 0.2426, 0.2400, 0.0026

0, 0.40, 0.2281, 0.2256, 0.0025

0, 0.42, 0.2146, 0.2122, 0.0024

0, 0.44, 0.2021, 0.1998, 0.0023

0, 0.46, 0.1905, 0.1882, 0.0023

0, 0.48, 0.1797, 0.1774, 0.0023

0, 0.50, 0.1696, 0.1673, 0.0023

0, 0.52, 0.1602, 0.1580, 0.0022

0, 0.54, 0.1514, 0.1493, 0.0021

0, 0.56, 0.1432, 0.1412, 0.0020

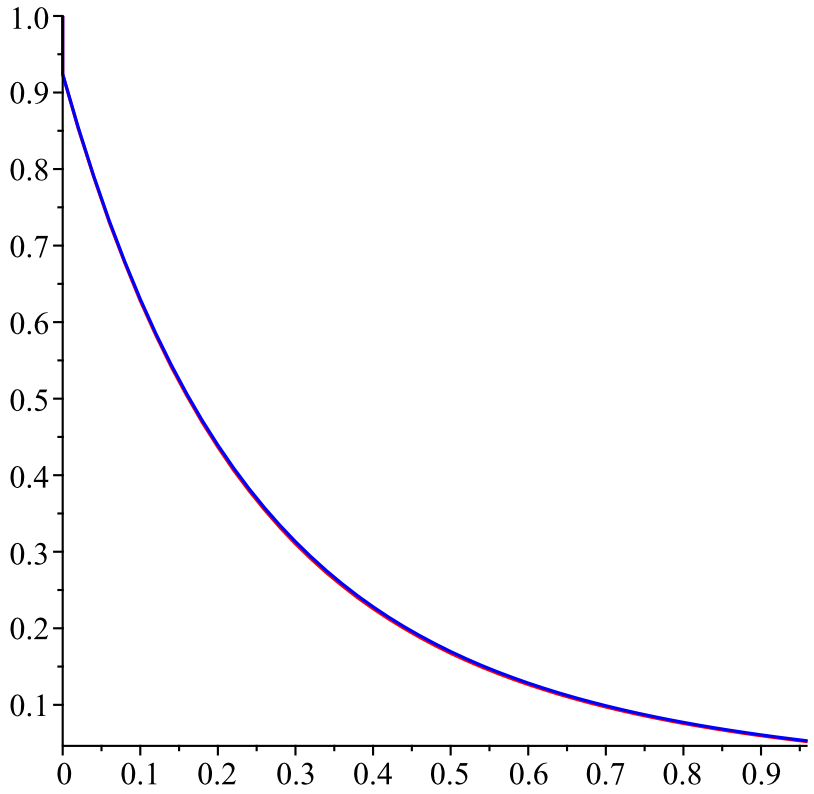
0, 0.58, 0.1356, 0.1336, 0.0020

0, 0.60, 0.1284, 0.1265, 0.0019

0, 0.62, 0.1217, 0.1199, 0.0018

0, 0.64, 0.1154, 0.1136, 0.0018

0, 0.66, 0.1095, 0.1078, 0.0017
0, 0.68, 0.1040, 0.1023, 0.0017
0, 0.70, 0.09880, 0.09718, 0.00162
0, 0.72, 0.09392, 0.09235, 0.00157
0, 0.74, 0.08932, 0.08782, 0.00150
0, 0.76, 0.08499, 0.08353, 0.00146
0, 0.78, 0.08091, 0.07951, 0.00140
0, 0.80, 0.07706, 0.07568, 0.00138
0, 0.82, 0.07342, 0.07215, 0.00127
0, 0.84, 0.06999, 0.06876, 0.00123
0, 0.86, 0.06674, 0.06554, 0.00120
0, 0.88, 0.06367, 0.06252, 0.00115
0, 0.90, 0.06076, 0.05964, 0.00112
0, 0.92, 0.05800, 0.05693, 0.00107
0, 0.94, 0.05539, 0.05435, 0.00104
0, 0.96, 0.05291, 0.05194, 0.00097
0, 0.98, 0.05056, 0.04962, 0.00094
0, 1.00, 0.04833, 0.04742, 0.00091




```
> RungeKutta := proc(a, b, h, y0, f)
  local i;
  local n := (b - a)/h;
  local t := a;
  local w := y0;
  local yi := w;
  local e := 0;
  local wt;
  local ode := diff(y(x), x) = f;
  local ics := y(0) = y0;
  local exactSolution := rhs(dsolve({ode, ics}));
  local k1, k2, k3;
   local dataPointsY, dataPointsW;
  local xList := [t];
  local wList := [w];
   local yList := [yi];
```

```
    print(Step, t_i, w_i, y_i, e_i);
    print(0, t, w, yi, e);

    for i from 1 to n + 1 do:
    yi := evalf(eval(exactSolution, x = t + h));
    k1 := eval(f, [x = t, y = w]);
    k2 := eval(f, [x = t + (h/2), y = w + (h/2)·k1]);
    k3 := eval(f, [x = t + h, y = w − h·k1 + 2 h·k2]);

    w := evalf(w + (h/6)·(k1 + 4 k2 + k3));
    e := abs(yi − w);
    print(0, t, w, yi, e);

    xList := [op(xList), t];
    wList := [op(wList), w];
    yList := [op(yList), yi];
     t := t + h;


    end do:
    dataPointsY := plot([seq([xList[i], yList[i]], i = 1 ..n)], style = line, color = red):
    dataPointsW := plot([seq([xList[i], wList[i]], i = 1 ..n)], style = line, color = blue):
    plots[display]({dataPointsY, dataPointsW});

    end proc
RungeKutta := proc(a, b, h, y0, f)                                          (3)
    local i, n, t, w, yi, e, wt, ode, ics, exactSolution, k1, k2, k3, dataPointsY, dataPointsW, xList,
    wList, yList;
    n := (b − a)/h;
    t := a;
    w := y0;
    yi := w;
    e := 0;
    ode := diff(y(x), x) = f;
    ics := y(0) = y0;
    exactSolution := rhs(dsolve({ode, ics}));
    xList := [t];
    wList := [w];
    yList := [yi];
    print(Step, t[i], w[i], y[i], e[i]);
    print(0, t, w, yi, e);
    for i to n + 1 do
        yi := evalf(eval(exactSolution, x = t + h));
        k1 := eval(f, [x = t, y = w]);
        k2 := eval(f, [x = t + 1/2 * h, y = w + 1/2 * h * k1]);
```

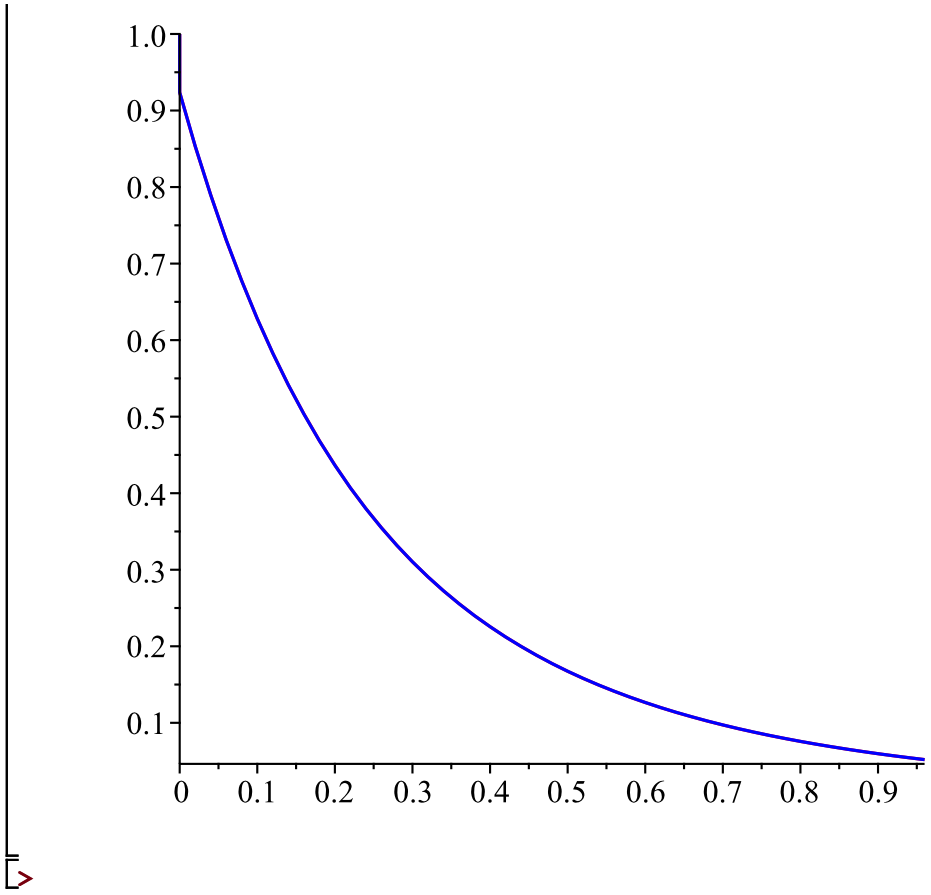
```
        k3 := eval(f, [x = t + h, y = w - h * k1 + 2 * h * k2]);
        w := evalf(w + 1/6 * h * (k1 + 4 * k2 + k3));
        e := abs(yi - w);
        print(0, t, w, yi, e);
        xList := [op(xList), t];
        wList := [op(wList), w];
        yList := [op(yList), yi];
        t := t + h
    end do;
    dataPointsY := plot([seq([xList[i], yList[i]], i = 1 ..n)], style = line, color = red);
    dataPointsW := plot([seq([xList[i], wList[i]], i = 1 ..n)], style = line, color = blue);
    plots[display]({dataPointsY, dataPointsW})
end proc
> Digits := 4; RungeKutta(0, 1, 0.02, 1, -5·y(x) + exp(-2·x));
```

$$Digits := 4$$

$$Step, t_i, w_i, y_i, e_i$$

$$0, 0, 1, 1, 0$$
$$0, 0, 0.9235, 0.9238, 0.0003$$
$$0, 0.02, 0.8535, 0.8531, 0.0004$$
$$0, 0.04, 0.7895, 0.7897, 0.0002$$
$$0, 0.06, 0.7309, 0.7306, 0.0003$$
$$0, 0.08, 0.6772, 0.6775, 0.0003$$
$$0, 0.10, 0.6280, 0.6278, 0.0002$$
$$0, 0.12, 0.5829, 0.5830, 0.0001$$
$$0, 0.14, 0.5415, 0.5414, 0.0001$$
$$0, 0.16, 0.5035, 0.5038, 0.0003$$
$$0, 0.18, 0.4686, 0.4687, 0.0001$$
$$0, 0.20, 0.4365, 0.4368, 0.0003$$
$$0, 0.22, 0.4070, 0.4069, 0.0001$$
$$0, 0.24, 0.3798, 0.3799, 0.0001$$
$$0, 0.26, 0.3548, 0.3549, 0.0001$$
$$0, 0.28, 0.3317, 0.3317, 0.$$
$$0, 0.30, 0.3104, 0.3103, 0.0001$$
$$0, 0.32, 0.2907, 0.2907, 0.$$
$$0, 0.34, 0.2725, 0.2724, 0.0001$$
$$0, 0.36, 0.2556, 0.2557, 0.0001$$
$$0, 0.38, 0.2400, 0.2400, 0.$$
$$0, 0.40, 0.2256, 0.2256, 0.$$
$$0, 0.42, 0.2122, 0.2122, 0.$$
$$0, 0.44, 0.1998, 0.1998, 0.$$
$$0, 0.46, 0.1882, 0.1882, 0.$$
$$0, 0.48, 0.1774, 0.1774, 0.$$

0, 0.50, 0.1674, 0.1673, 0.0001
0, 0.52, 0.1581, 0.1580, 0.0001
0, 0.54, 0.1494, 0.1493, 0.0001
0, 0.56, 0.1413, 0.1412, 0.0001
0, 0.58, 0.1337, 0.1336, 0.0001
0, 0.60, 0.1266, 0.1265, 0.0001
0, 0.62, 0.1199, 0.1199, 0.
0, 0.64, 0.1137, 0.1136, 0.0001
0, 0.66, 0.1079, 0.1078, 0.0001
0, 0.68, 0.1024, 0.1023, 0.0001
0, 0.70, 0.09726, 0.09718, 0.00008
0, 0.72, 0.09242, 0.09235, 0.00007
0, 0.74, 0.08787, 0.08782, 0.00005
0, 0.76, 0.08359, 0.08353, 0.00006
0, 0.78, 0.07956, 0.07951, 0.00005
0, 0.80, 0.07575, 0.07568, 0.00007
0, 0.82, 0.07216, 0.07215, 0.00001
0, 0.84, 0.06877, 0.06876, 0.00001
0, 0.86, 0.06556, 0.06554, 0.00002
0, 0.88, 0.06253, 0.06252, 0.00001
0, 0.90, 0.05966, 0.05964, 0.00002
0, 0.92, 0.05694, 0.05693, 0.00001
0, 0.94, 0.05437, 0.05435, 0.00002
0, 0.96, 0.05193, 0.05194, 0.00001
0, 0.98, 0.04962, 0.04962, 0.
0, 1.00, 0.04742, 0.04742, 0.


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

```
> EulerSystem := proc(a, b, h, x0, fx, y0, fy)
  local wxi, wyi, dataPointsYx, dataPointsWx, dataPointsYy, dataPointsWy;
  local i;
  local n := (b - a)/h;
  local t := a;
  local wx := x0;
  local wy := y0;

  local yix := wx;
  local yiy := wy;

  local ex := 0;
  local ey := 0;


  local ode := {diff(x1(x), x) = fx, diff(x2(x), x) = fy};
  local ics := {x1(0) = x0, x2(0) = y0};
  local exactSolutionX := rhs(dsolve({diff(x1(x), x) = fx, diff(x2(x), x) = fy, x1(0) = x0, x2(0)
      = y0})[1]); local exactSolutionY := rhs(dsolve({diff(x1(x), x) = fx, diff(x2(x), x) = fy, x1(0)
      = x0, x2(0) = y0})[2]);


  local dataPointsY, dataPointsW;
  local xList := [t];
  local wListX := [wx];
  local yListX := [yix];
  local wListY := [wy];
  local yListY := [yiy];


  print(Step, t[i], wx[i], yx[i], ex[i], wy[i], yy[i], ey[i]);
  print(0, t, wx, yix, ex, wy, yiy, ey);


  for i from 1 to n + 1 do:
  yix := evalf(eval(exactSolutionX, x = t + h));
  yiy := evalf(eval(exactSolutionY, x = t + h));

  wxi := wx + h*evalf(eval(fx, [x1 = wx, x2 = wy]));
  wyi := wy + h*evalf(eval(fy, [x1 = wx, x2 = wy]));
  wx := wxi;
  wy := wyi;
```

```
    ex := abs(yix − wx);
    ey := abs(yiy − wy);

    print(0, t, wx, yix, ex, wy, yiy, ey);

     xList := [op(xList), t];
    wListX := [op(wListX), wx];
    yListX := [op(yListX), yix];
     wListY := [op(wListY), wy];
    yListY := [op(yListY), yiy]; t := t + h;


    end do:
   dataPointsYx := plot([seq([xList[i], yListX[i]], i = 1 ..n)], style = line, color = red) :
    dataPointsWx := plot([seq([xList[i], wListX[i]], i = 1 ..n)], style = line, color = blue) :
    dataPointsYy := plot([seq([xList[i], yListY[i]], i = 1 ..n)], style = line, color = red) :
    dataPointsWy := plot([seq([xList[i], wListY[i]], i = 1 ..n)], style = line, color = blue) :

   plots[display]({dataPointsYx, dataPointsWx, dataPointsYy, dataPointsWy});


    end proc
EulerSystem := proc(a, b, h, x0, fx, y0, fy)                                    (1)
    local wxi, wyi, dataPointsYx, dataPointsWx, dataPointsYy, dataPointsWy, i, n, t, wx, wy, yix,
    yiy, ex, ey, ode, ics, exactSolutionX, exactSolutionY, dataPointsY, dataPointsW, xList,
    wListX, yListX, wListY, yListY;
    n := (b − a) / h;
    t := a;
    wx := x0;
    wy := y0;
    yix := wx;
    yiy := wy;
    ex := 0;
    ey := 0;
    ode := {diff(x[1](x), x) = fx, diff(x[2](x), x) = fy};
    ics := {x[1](0) = x0, x[2](0) = y0};
    exactSolutionX := rhs(dsolve([diff(x[1](x), x) = fx, diff(x[2](x), x) = fy, x[1](0) = x0, x
    [2](0) = y0])[1]);
    exactSolutionY := rhs(dsolve([diff(x[1](x), x) = fx, diff(x[2](x), x) = fy, x[1](0) = x0, x
    [2](0) = y0])[2]);
    xList := [t];
    wListX := [wx];
    yListX := [yix];
    wListY := [wy];
    yListY := [yiy];
    print(Step, t[i], wx[i], yx[i], ex[i], wy[i], yy[i], ey[i]);
```

```
    print(0, t, wx, yix, ex, wy, yiy, ey);
    for i to n + 1 do
        yix := evalf(eval(exactSolutionX, x = t + h));
        yiy := evalf(eval(exactSolutionY, x = t + h));
        wxi := wx + h * evalf(eval(fx, [x[1] = wx, x[2] = wy]));
        wyi := wy + h * evalf(eval(fy, [x[1] = wx, x[2] = wy]));
        wx := wxi;
        wy := wyi;
        ex := abs(yix - wx);
        ey := abs(yiy - wy);
        print(0, t, wx, yix, ex, wy, yiy, ey);
        xList := [op(xList), t];
        wListX := [op(wListX), wx];
        yListX := [op(yListX), yix];
        wListY := [op(wListY), wy];
        yListY := [op(yListY), yiy];
        t := t + h
    end do;
    dataPointsYx := plot([seq([xList[i], yListX[i]], i = 1 ..n)], style = line, color = red);
    dataPointsWx := plot([seq([xList[i], wListX[i]], i = 1 ..n)], style = line, color = blue);
    dataPointsYy := plot([seq([xList[i], yListY[i]], i = 1 ..n)], style = line, color = red);
    dataPointsWy := plot([seq([xList[i], wListY[i]], i = 1 ..n)], style = line, color = blue);
    plots[display]({dataPointsYx, dataPointsWx, dataPointsYy, dataPointsWy})
end proc
```

```
> Digits := 4; EulerSystem(0, 1, 0.02, 5, -7 x1(x) + 8 x2(x), 0, -6 x1(x) + 7 x2(x));
```

$$Digits := 4$$

$$Step,\ t_i,\ wx_i,\ yx_i,\ ex_i,\ wy_i,\ yy_i,\ ey_i$$

$$0,\ 0,\ 5,\ 5,\ 0,\ 0,\ 0,\ 0$$

$$0,\ 0,\ 4.30,\ 4.30,\ 0.,\ -0.60,\ -0.60,\ 0.$$

$$0,\ 0.02,\ 3.602,\ 3.60,\ 0.002,\ -1.200,\ -1.21,\ 0.010$$

$$0,\ 0.04,\ 2.906,\ 2.91,\ 0.004,\ -1.800,\ -1.80,\ 0.$$

$$0,\ 0.06,\ 2.211,\ 2.22,\ 0.009,\ -2.401,\ -2.39,\ 0.011$$

$$0,\ 0.08,\ 1.517,\ 1.52,\ 0.003,\ -3.003,\ -3.01,\ 0.007$$

$$0,\ 0.10,\ 0.8242,\ 0.84,\ 0.0158,\ -3.605,\ -3.60,\ 0.005$$

$$0,\ 0.12,\ 0.1320,\ 0.14,\ 0.0080,\ -4.209,\ -4.21,\ 0.001$$

$$0,\ 0.14,\ -0.5598,\ -0.57,\ 0.0102,\ -4.814,\ -4.83,\ 0.016$$

$$0,\ 0.16,\ -1.252,\ -1.25,\ 0.002,\ -5.421,\ -5.43,\ 0.009$$

$$0,\ 0.18,\ -1.944,\ -1.95,\ 0.006,\ -6.030,\ -6.04,\ 0.010$$

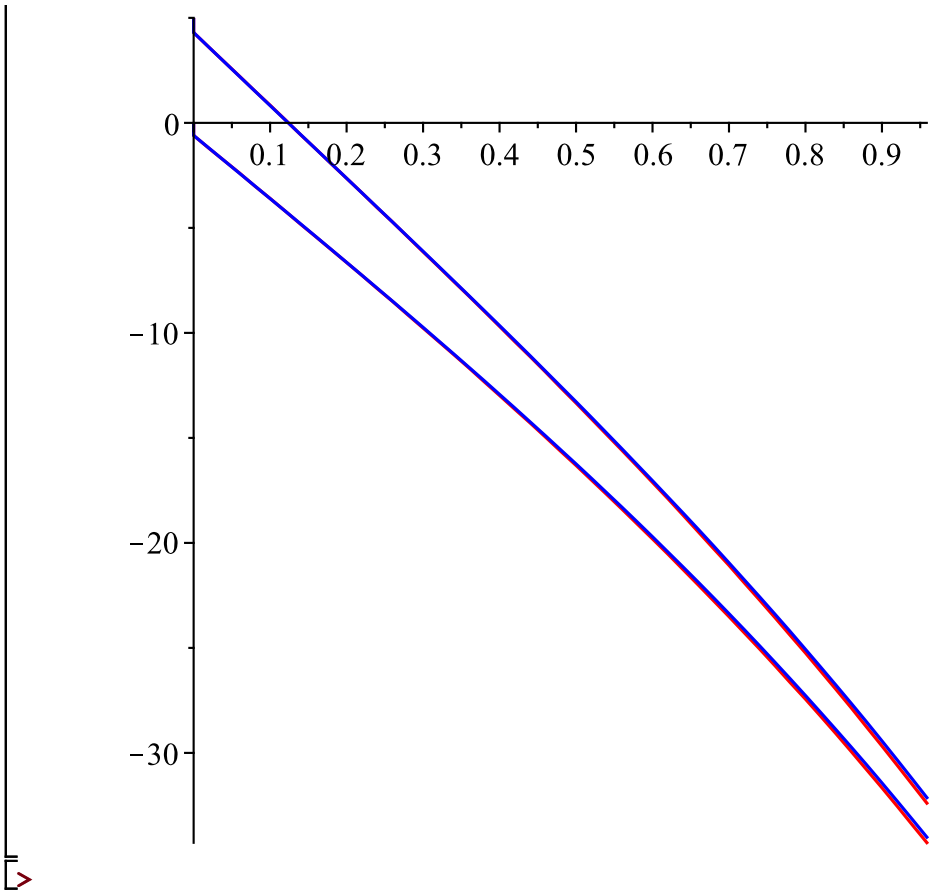
$$0,\ 0.20,\ -2.637,\ -2.64,\ 0.003,\ -6.641,\ -6.65,\ 0.009$$

$$0,\ 0.22,\ -3.330,\ -3.33,\ 0.,\ -7.254,\ -7.26,\ 0.006$$

$$0,\ 0.24,\ -4.024,\ -4.04,\ 0.016,\ -7.870,\ -7.89,\ 0.020$$

$$0,\ 0.26,\ -4.720,\ -4.72,\ 0.,\ -8.489,\ -8.50,\ 0.011$$

0, 0.28, −5.417, −5.43, 0.013, −9.111, −9.14, 0.029
0, 0.30, −6.116, −6.14, 0.024, −9.737, −9.77, 0.033
0, 0.32, −6.818, −6.84, 0.022, −10.37, −10.40, 0.03
0, 0.34, −7.523, −7.55, 0.027, −11.00, −11.03, 0.03
0, 0.36, −8.230, −8.25, 0.020, −11.64, −11.67, 0.03
0, 0.38, −8.940, −8.97, 0.030, −12.28, −12.33, 0.05
0, 0.40, −9.653, −9.69, 0.037, −12.93, −12.98, 0.05
0, 0.42, −10.37, −10.42, 0.05, −13.58, −13.64, 0.06
0, 0.44, −11.09, −11.13, 0.04, −14.24, −14.29, 0.05
0, 0.46, −11.82, −11.86, 0.04, −14.90, −14.96, 0.06
0, 0.48, −12.55, −12.61, 0.06, −15.57, −15.64, 0.07
0, 0.50, −13.28, −13.34, 0.06, −16.24, −16.31, 0.07
0, 0.52, −14.02, −14.09, 0.07, −16.92, −17.00, 0.08
0, 0.54, −14.77, −14.84, 0.07, −17.61, −17.69, 0.08
0, 0.56, −15.52, −15.59, 0.07, −18.30, −18.39, 0.09
0, 0.58, −16.28, −16.35, 0.07, −19.00, −19.10, 0.10
0, 0.60, −17.04, −17.12, 0.08, −19.71, −19.81, 0.10
0, 0.62, −17.81, −17.89, 0.08, −20.43, −20.53, 0.10
0, 0.64, −18.58, −18.68, 0.10, −21.15, −21.27, 0.12
0, 0.66, −19.36, −19.48, 0.12, −21.88, −22.01, 0.13
0, 0.68, −20.15, −20.28, 0.13, −22.62, −22.76, 0.14
0, 0.70, −20.95, −21.07, 0.12, −23.37, −23.51, 0.14
0, 0.72, −21.76, −21.90, 0.14, −24.13, −24.28, 0.15
0, 0.74, −22.57, −22.72, 0.15, −24.90, −25.05, 0.15
0, 0.76, −23.39, −23.55, 0.16, −25.68, −25.84, 0.16
0, 0.78, −24.22, −24.40, 0.18, −26.47, −26.65, 0.18
0, 0.80, −25.07, −25.24, 0.17, −27.27, −27.44, 0.17
0, 0.82, −25.92, −26.11, 0.19, −28.08, −28.26, 0.18
0, 0.84, −26.78, −26.98, 0.20, −28.90, −29.09, 0.19
0, 0.86, −27.65, −27.86, 0.21, −29.73, −29.94, 0.21
0, 0.88, −28.53, −28.77, 0.24, −30.57, −30.80, 0.23
0, 0.90, −29.43, −29.67, 0.24, −31.43, −31.66, 0.23
0, 0.92, −30.34, −30.59, 0.25, −32.30, −32.54, 0.24
0, 0.94, −31.26, −31.52, 0.26, −33.18, −33.44, 0.26
0, 0.96, −32.19, −32.45, 0.26, −34.07, −34.33, 0.26
0, 0.98, −33.14, −33.41, 0.27, −34.98, −35.25, 0.27
0, 1.00, −34.10, −34.39, 0.29, −35.90, −36.19, 0.29


0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
−10
−20
−30

12 Numerical Solution of ODEs, BVPs

Write Maple/Matlab code to solve the following BVP:

$$\begin{cases} y''(t) = 4y(t) \\ y(0) = 1 \\ y(1) = 3 \end{cases}$$

using central differences, stepsize $h = 0.01$ and 4 digits of accuracy. Show clearly all steps of the solution process. Plot the matrix of the linear system using Maple's matrix browser tool, to visualize the tridiagonal structure of the matrix. Solve the resulting tridiagonal linear system by LU decomposition. Show clearly the table of values as in the previous chapter on IVPs. Plot the exact solution against the numerical solutions found by your code.

Answer:
As seen from a portion of the exported Matrix below, it is very clear that the Matrix A that was generated from the system of discretized linear equations was tri-diagonal. The matrix was solved using LU Decomposition and the results seen below in the following pages of Maple code.

Snippet.png

Step 1: Mesh

```
> eq := diff(y(t),t$2) - 4*y(t) = 0:
  a := 0:
  y_a := 1:
  b := 1:
  y_b := 3:
  h := 0.01:
  n := trunc((b-a)/h);
  Digits := 4:
```

$$n := 100 \quad \textbf{(1)}$$

Step 2: Discretization for the BVP

```
> yList := Array([seq(y[i], i=0..n)]):
  yList[1] := y_a:
  yList[n+1] := y_b:
  equations := []:

  for i from 1 to n-1 do
  equations := [op(equations,), eval(eq, [diff(y(t),t$2)
      = (yList[i] - 2*yList[i+1] + yList[i+2])/h^2, y(t) = yList[i+1]])]:
  end do:
```

Step 3: Obtain Tri-Diagonal Matrix and Solve for System of Equations

```
> List := [seq(y[i], i=1..n-1)]:
  with(LinearAlgebra):
  A, colY := GenerateMatrix(equations, List):
  A;
  answers := LinearSolve(A, colY, method='LU');
```

$$\begin{bmatrix} \textit{99 x 99 Matrix} \\ \textit{Data Type: anything} \\ \textit{Storage: rectangular} \\ \textit{Order: Fortran\_order} \end{bmatrix}$$

(2)

$$answers := \begin{bmatrix} 1 \,..\, 99\ Vector_{column} \\ Data\ Type: float_8 \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \quad \textbf{(2)}$$

Step 4: Solve for exact solution to the DVP and Display Results

```
> exactSolution := rhs(dsolve({diff(y(x),x$2) - 4*y(x) = 0, y(a) = y_a, y(b) = y_b}));
  xVal := a :
  printResults := [[Step, x_i, Approximate Solution, Exact Solution, Error], [0, a, y_a, y_a, 0]] :
  exactSolutions := [y_a] :
  xValues := [a] :
  approxSolutions := [y_a] :

  print(Step, x_i, Approximate Solution, Exact Solution, Error);
  print(0, a, y_a, y_a, 0);

  for i from 1 to n - 1 do:
  xVal := xVal + h;
  exactAns := evalf(eval(exactSolution, x = xVal));
  err := evalf(abs(exactAns - answers[i]));
  printResults := [op(printResults), [i, xVal, evalf(answers[i]), exactAns, err]];
  print(i, xVal, evalf(answers[i]), exactAns, err);
  exactSolutions := [op(exactSolutions), exactAns] :
  xValues := [op(xValues), xVal] :
  approxSolutions := [op(approxSolutions), answers[i]] :

  end do:
  printResults := [op(printResults), [n, b, y_b, y_b, 0]] :

  print(n, b, y_b, y_b, 0);


   exactSolutions := [op(exactSolutions), y_b] :
  xValues := [op(xValues), b] :
  approxSolutions := [op(approxSolutions), y_b] :


  dataPointsApprox := plot([seq([xValues[i], approxSolutions[i]], i = 1 ..n + 1)], style = line,
      color = red) :
  dataPointsExact := plot([seq([xValues[i], exactSolutions[i]], i = 1 ..n + 1)], style = line, color
      = blue) :
  plots[display]({dataPointsApprox, dataPointsExact})
```

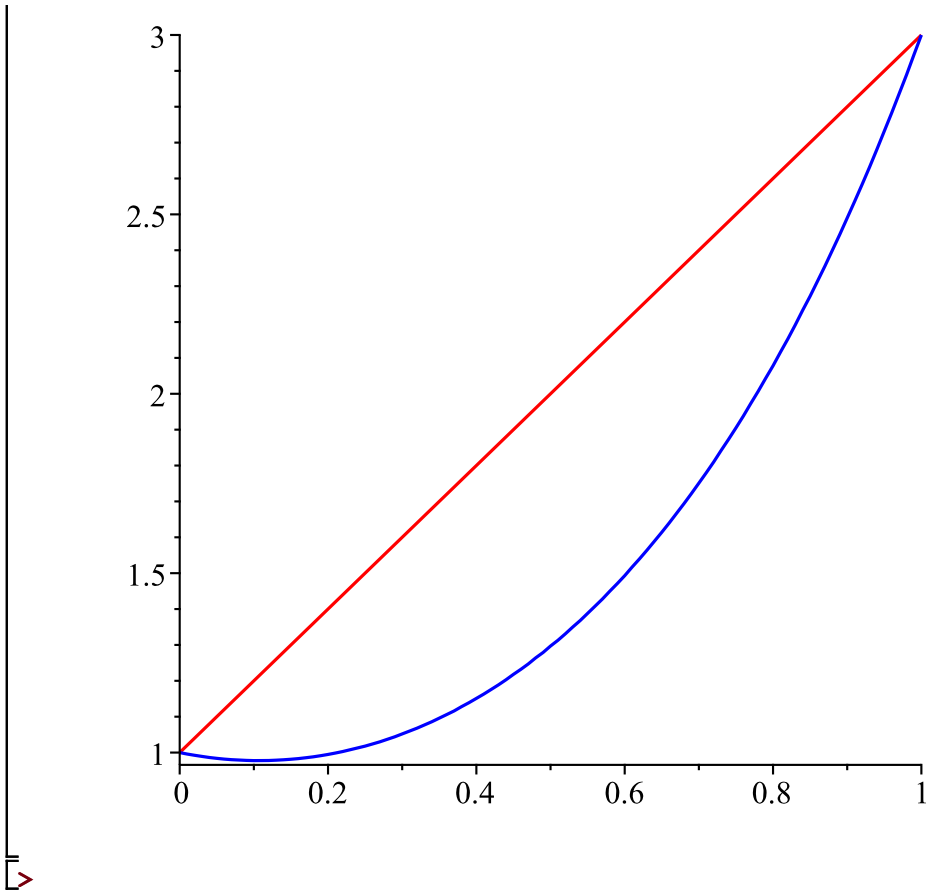
$$exactSolution := -\frac{\left(e^2-3\right)\,e^{-2x}}{e^{-2}-e^2}+\frac{\left(-3+e^{-2}\right)\,e^{2x}}{e^{-2}-e^2}$$

Step, x_i, *Approximate Solution*, *Exact Solution*, *Error*

0, 0, 1, 1, 0
1, 0.01, 1.020, 0.9959, 0.02410
2, 0.02, 1.040, 0.9925, 0.04750
3, 0.03, 1.060, 0.9893, 0.07070
4, 0.04, 1.080, 0.9863, 0.09370
5, 0.05, 1.100, 0.9839, 0.1161
6, 0.06, 1.120, 0.9818, 0.1382
7, 0.07, 1.140, 0.9802, 0.1598
8, 0.08, 1.160, 0.9792, 0.1808
9, 0.09, 1.180, 0.9782, 0.2018
10, 0.10, 1.200, 0.9776, 0.2224
11, 0.11, 1.220, 0.9777, 0.2423
12, 0.12, 1.240, 0.9779, 0.2621
13, 0.13, 1.260, 0.9788, 0.2812
14, 0.14, 1.280, 0.9799, 0.3001
15, 0.15, 1.300, 0.9814, 0.3186
16, 0.16, 1.320, 0.9832, 0.3368
17, 0.17, 1.340, 0.9856, 0.3544
18, 0.18, 1.360, 0.9881, 0.3719
19, 0.19, 1.380, 0.9913, 0.3887
20, 0.20, 1.400, 0.9948, 0.4052
21, 0.21, 1.420, 0.9987, 0.4213
22, 0.22, 1.440, 1.003, 0.4370
23, 0.23, 1.460, 1.008, 0.4520
24, 0.24, 1.480, 1.013, 0.4670
25, 0.25, 1.500, 1.018, 0.4820
26, 0.26, 1.520, 1.024, 0.4960
27, 0.27, 1.540, 1.030, 0.5100
28, 0.28, 1.560, 1.037, 0.5230
29, 0.29, 1.580, 1.044, 0.5360
30, 0.30, 1.600, 1.052, 0.5480
31, 0.31, 1.620, 1.060, 0.5600
32, 0.32, 1.640, 1.068, 0.5720
33, 0.33, 1.660, 1.077, 0.5830
34, 0.34, 1.680, 1.086, 0.5940
35, 0.35, 1.700, 1.096, 0.6040
36, 0.36, 1.720, 1.106, 0.6140
37, 0.37, 1.740, 1.116, 0.6240
38, 0.38, 1.760, 1.128, 0.6320

39, 0.39, 1.780, 1.139, 0.6410
40, 0.40, 1.800, 1.151, 0.6490
41, 0.41, 1.820, 1.163, 0.6570
42, 0.42, 1.840, 1.176, 0.6640
43, 0.43, 1.860, 1.189, 0.6710
44, 0.44, 1.880, 1.203, 0.6770
45, 0.45, 1.900, 1.218, 0.6820
46, 0.46, 1.920, 1.232, 0.6880
47, 0.47, 1.940, 1.247, 0.6930
48, 0.48, 1.960, 1.264, 0.6960
49, 0.49, 1.980, 1.279, 0.7010
50, 0.50, 2.000, 1.297, 0.7030
51, 0.51, 2.020, 1.313, 0.7070
52, 0.52, 2.040, 1.331, 0.7090
53, 0.53, 2.060, 1.350, 0.7100
54, 0.54, 2.080, 1.368, 0.7120
55, 0.55, 2.100, 1.388, 0.7120
56, 0.56, 2.120, 1.408, 0.7120
57, 0.57, 2.140, 1.428, 0.7120
58, 0.58, 2.160, 1.450, 0.7100
59, 0.59, 2.180, 1.471, 0.7090
60, 0.60, 2.200, 1.493, 0.7070
61, 0.61, 2.220, 1.517, 0.7030
62, 0.62, 2.240, 1.540, 0.7000
63, 0.63, 2.260, 1.564, 0.6960
64, 0.64, 2.280, 1.589, 0.6910
65, 0.65, 2.300, 1.614, 0.6860
66, 0.66, 2.320, 1.640, 0.6800
67, 0.67, 2.340, 1.667, 0.6730
68, 0.68, 2.360, 1.694, 0.6660
69, 0.69, 2.380, 1.722, 0.6580
70, 0.70, 2.400, 1.751, 0.6490
71, 0.71, 2.420, 1.780, 0.6400
72, 0.72, 2.440, 1.810, 0.6300
73, 0.73, 2.460, 1.842, 0.6180
74, 0.74, 2.480, 1.873, 0.6070
75, 0.75, 2.500, 1.905, 0.5950
76, 0.76, 2.520, 1.938, 0.5820
77, 0.77, 2.540, 1.973, 0.5670
78, 0.78, 2.560, 2.007, 0.5530
79, 0.79, 2.580, 2.043, 0.5370
80, 0.80, 2.600, 2.078, 0.5220

81, 0.81, 2.620, 2.116, 0.5040
82, 0.82, 2.640, 2.153, 0.4870
83, 0.83, 2.660, 2.192, 0.4680
84, 0.84, 2.680, 2.233, 0.4470
85, 0.85, 2.700, 2.272, 0.4280
86, 0.86, 2.720, 2.314, 0.4060
87, 0.87, 2.740, 2.356, 0.3840
88, 0.88, 2.760, 2.400, 0.3600
89, 0.89, 2.780, 2.444, 0.3360
90, 0.90, 2.800, 2.490, 0.3100
91, 0.91, 2.820, 2.536, 0.2840
92, 0.92, 2.840, 2.583, 0.2570
93, 0.93, 2.860, 2.631, 0.2290
94, 0.94, 2.880, 2.681, 0.1990
95, 0.95, 2.900, 2.732, 0.1680
96, 0.96, 2.920, 2.783, 0.1370
97, 0.97, 2.940, 2.836, 0.1040
98, 0.98, 2.960, 2.889, 0.07100
99, 0.99, 2.980, 2.945, 0.03500
100, 1, 3, 3, 0


3
2.5
2
1.5
1
0
0.2
0.4
0.6
0.8
1

13 Numerical Solution of PDEs, Finite Differences

Adapt the 2D Laplace equation discretization Maple code provided and discussed in class, to discretize and solve the following 2D Laplace PDE (with non-constant boundary conditions):

$$\begin{cases} u_{xx}(x,y) + u_{yy}(x,y) = 0 \\ u(x,0) = x^2 \\ u(x,1) = x^2 - 1 \\ u(0,y) = -y^2 \\ u(1,y) = 1 - y^2 \end{cases}$$

(1) Use the stepsize $h = 0.1$ along the x and y axes, to discretize and solve the above PDE.

(2) Use the stepsize $h = 0.01$ along the x and y axes, to discretize and solve the above PDE.

Answer the following questions:

- How many interior nodes does the grid contain for $h = 0.1$?
- How many interior nodes does the grid contain for $h = 0.01$?
- How many equations and unknowns does the linear system contain for $h = 0.1$?
- How many equations and unknowns does the linear system contain for $h = 0.01$?
- Plot the matrix of the resulting linear system, for $h = 0.1$, to exhibit/visualize its structure
- Plot the matrix of the resulting linear system, for $h = 0.01$, to exhibit/visualize its structure
- Comment on the structure of the matrix of the resulting linear system, for $h = 0.1$
- Comment on the structure of the matrix of the resulting linear system, for $h = 0.01$
- Plot the approximate solution that you found, for $h = 0.1$ against the exact solution $u(x,y) = x^2 - y^2$
- Plot the approximate solution that you found, for $h = 0.01$ against the exact solution $u(x,y) = x^2 - y^2$

Answer:

- Grid contains 81 grid points for h=0.1
- Grid contains 9801 grid points for h=0.01
- Linear System contains 81 equations and 81 variables for h=0.01
- Linear System contains 9801 equations and 9801 variables for h=0.01
- See below for a snippet of the matrix of the resulting linear system for h=0.1 in Maple Code
- See below for a snippet of the matrix of the resulting linear system for h=0.01 in Maple Code
- The matrix for h=0.1 is penta-diagonal and block-tridiagonal
- The matrix for h=0.01 is penta-diagonal and block-tridiagonal
- See following pages of maple code for plots of approximate solutions (red) in comparison with exact solutions (blue). From these plots it is evident that the approximation is very good for h=0.1 and h=0.01 because the graphs overlap.

```
> restart;
```

sample Maple code for discretization of the 2D Laplace PDE

parameter initialization

```
> a := 0;
```

$$a := 0 \tag{1}$$

```
> b := 1;
```

$$b := 1 \tag{2}$$

```
>
```

run this code with n=6, n=8, n=10 etc

```
> n := 10;
```

$$n := 10 \tag{3}$$

```
> h := evalf((b-a)/n);
```

$$h := 0.1000000000 \tag{4}$$

definition of the subdivision points on the x-axis and y-axis

```
> x := [seq(a+i*h,i=0..n)]:
  nops(x);
```

$$11 \tag{5}$$

```
> y := [seq(a+i*h,i=0..n)]:
  nops(y);
```

$$11 \tag{6}$$

```
> # total grid points
  nops(x)*nops(y);
  # total interior grid points
  (nops(x)-2)*(nops(y)-2);
```

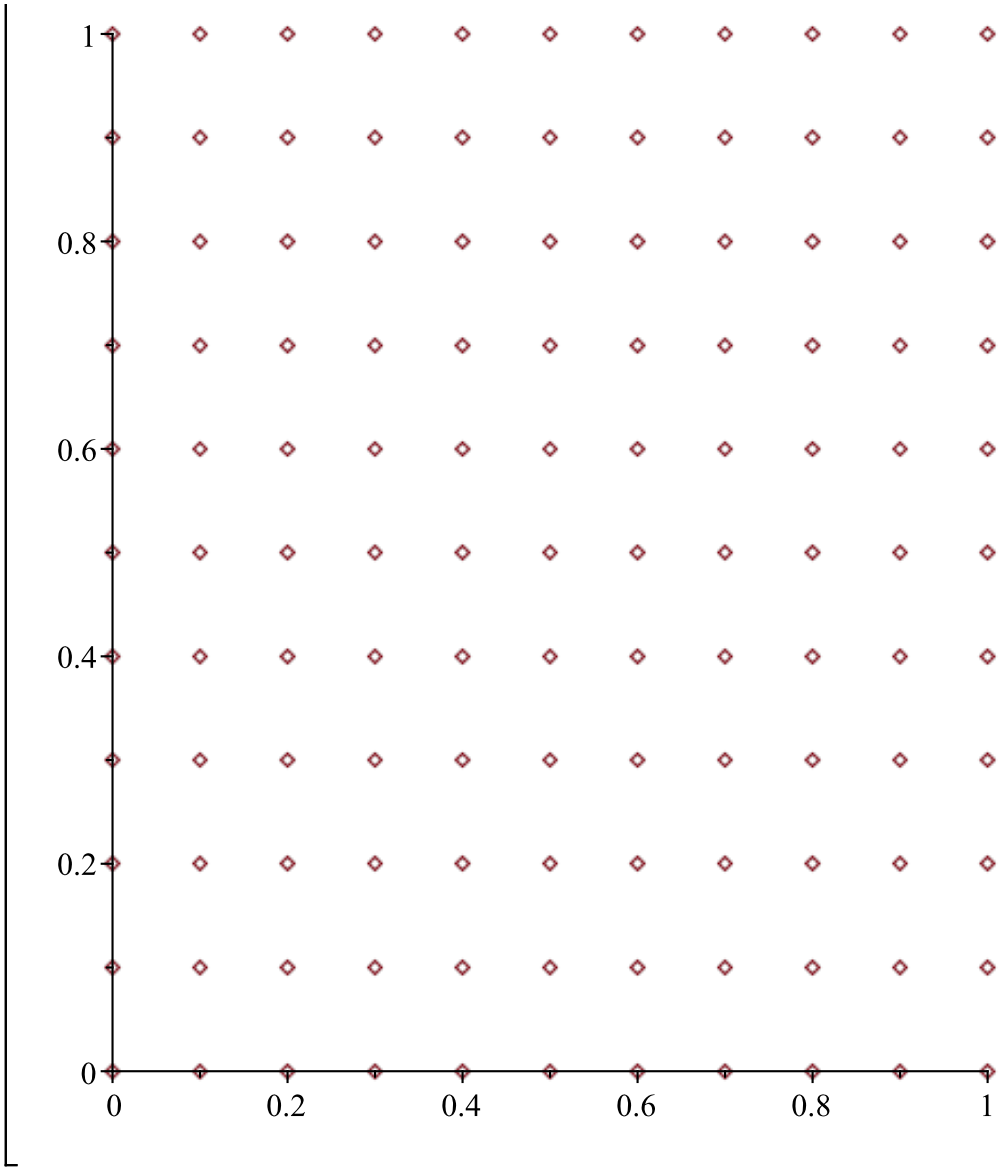
$$121$$
$$81 \tag{7}$$

definition of the grid points

```
> grid := [];
```

$$grid := [\,] \tag{8}$$

```
> for i from 1 to nops(x) do
  for j from 1 to nops(y) do
     grid := [op(grid),[x[i],y[j]]]:
  end do
  end do
> grid:
  nops(grid):
> plot(grid,style=point);
```



set up a 2D array to hold the values of the grid points

```
> uu := Array(0..n,0..n,(i,j)->u[i,j]):
```

initialize bottom grid points (excluding the two ends)

```
> for i from 1 to n-1 do uu[i,0] := (a+i*h)^2: end do:
```

initialize top grid points (excluding the two ends)

```
> for i from 1 to n-1 do uu[i,n] := (a+i*h)^2-1: end do:
```

initialize left grid points (excluding the two ends)

```
> for j from 1 to n-1 do uu[0,j] := -(a+j*h)^2: end do:
```

initialize right grid points (excluding the two ends)

```
> for j from 1 to n-1 do uu[n,j] := 1-(a+j*h)^2: end do:
```

set up the linear equations using the Laplace stencil, for the interior grid points, using the lexicographical ordering

```
> eqs := []:
  for i from 1 to n-1 do
  for j from 1 to n-1 do
       eqs := [op(eqs),uu[i+1,j]+uu[i-1,j]+uu[i,j+1]+uu[i,j-1]-4*uu
  [i,j]=0]:
  end do;
  end do;
```

```
> eqs:
  nops(eqs);
```

$$81 \tag{9}$$

```
> vars := [seq(seq(u[i,j],j=1..n-1),i=1..n-1)]:
  nops(vars);
```

$$81 \tag{10}$$

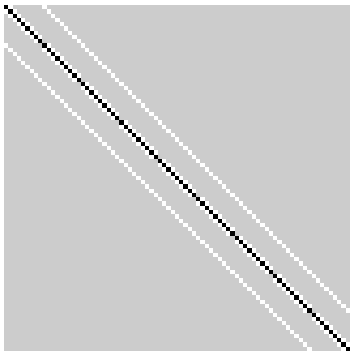
define the matrix and the constant vector for the above linear system of equations

```
> (A, b) := LinearAlgebra[GenerateMatrix]( eqs, vars ):
```

use Maple's Matrix Browser to visualize the structure of A (double click on the summary output below)

```
> A;
```

$$\begin{bmatrix} \textit{81 x 81 Matrix} \\ \textit{Data Type: anything} \\ \textit{Storage: rectangular} \\ \textit{Order: Fortran\_order} \end{bmatrix} \tag{11}$$



```
> approxSolutions:=LinearAlgebra[LinearSolve](A,b);
```

$$approxSolutions := \begin{bmatrix} 1 .. 81\ Vector_{column} \\ Data\ Type: float_8 \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \quad (12)$$

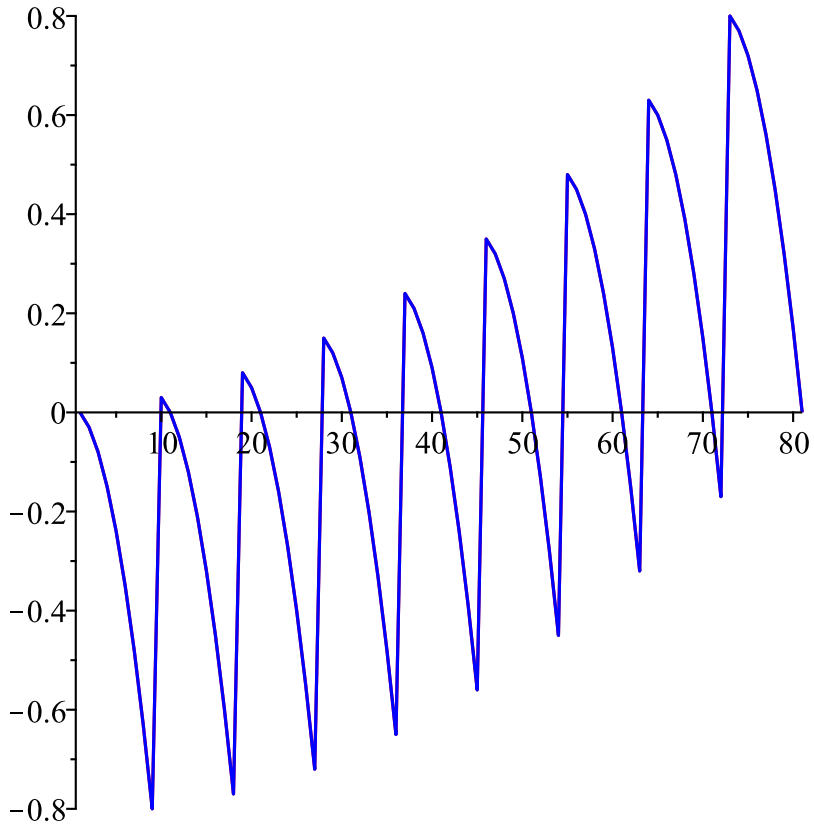
```
> exact := x_e^2 - y_e^2;
```

$$exact := x_e^2 - y_e^2 \quad (13)$$

```
> exactSolutions := [ ]:
  for i from 1 to n - 1 do:
   for j from 1 to n - 1 do:
    exactSolutions := [op(exactSolutions), evalf(eval(exact, [x_e = a + i·h, y_e = a + j·h]))]
  end do:
  end do:
  m := numelems(exactSolutions):
```

```
> dataPointsApprox := plot([seq([i, evalf(approxSolutions[i])], i = 1 ..m)], style = line, color
      = red): dataPointsExact := plot([seq([i, exactSolutions[i]], i = 1 ..m)], style = line, color
      = blue):

  plots[display]({dataPointsApprox, dataPointsExact})
```



The above graph displays the exact solution in blue and approximate solutions in red. If no red is seen then the approximation give almost the same result as the exact solution. The x-axis simply represents the sequence of points and y-axis represents the actual values.

```
> restart;
```

sample Maple code for discretization of the 2D Laplace PDE

parameter initialization

```
> a := 0;
```

$$a := 0 \tag{1}$$

```
> b := 1;
```

$$b := 1 \tag{2}$$

```
>
```

run this code with n=6, n=8, n=10 etc

```
> n := 100;
```

$$n := 100 \tag{3}$$

```
> h := evalf((b-a)/n);
```

$$h := 0.01000000000 \tag{4}$$

definition of the subdivision points on the x-axis and y-axis

```
> x := [seq(a+i*h,i=0..n)]:
  nops(x);
```

$$101 \tag{5}$$

```
> y := [seq(a+i*h,i=0..n)]:
  nops(y);
```

$$101 \tag{6}$$

```
> # total grid points
  nops(x)*nops(y);
  # total interior grid points
  (nops(x)-2)*(nops(y)-2);
```

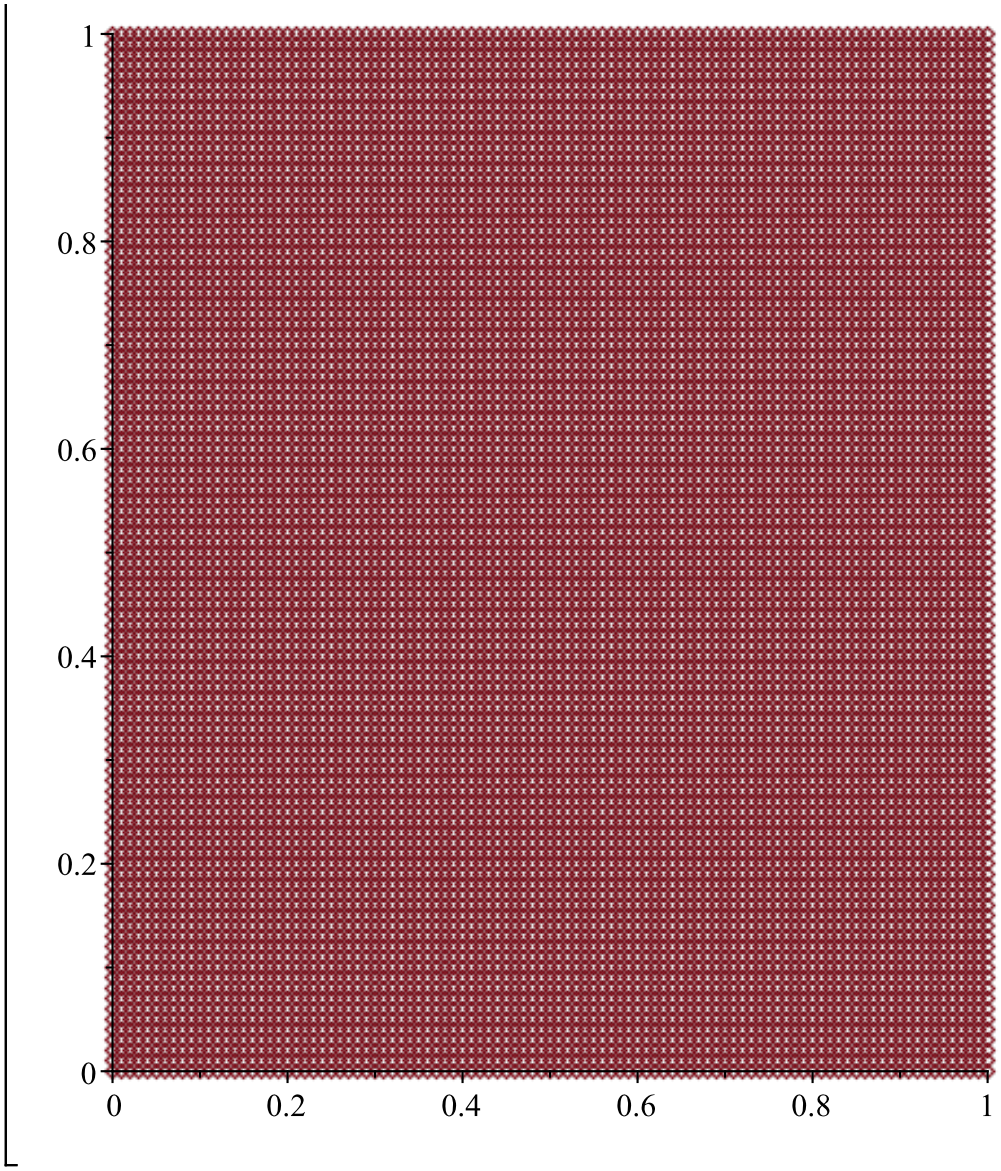
$$10201$$
$$9801 \tag{7}$$

definition of the grid points

```
> grid := [];
```

$$grid := [\,] \tag{8}$$

```
> for i from 1 to nops(x) do
  for j from 1 to nops(y) do
     grid := [op(grid),[x[i],y[j]]]:
  end do
  end do
> grid:
  nops(grid):
> plot(grid,style=point);
```



set up a 2D array to hold the values of the grid points

```
> uu := Array(0..n,0..n,(i,j)->u[i,j]):
```

initialize bottom grid points (excluding the two ends)

```
> for i from 1 to n-1 do uu[i,0] := (a+i*h)^2: end do:
```

initialize top grid points (excluding the two ends)

```
> for i from 1 to n-1 do uu[i,n] := (a+i*h)^2-1: end do:
```

initialize left grid points (excluding the two ends)

```
> for j from 1 to n-1 do uu[0,j] := -(a+j*h)^2: end do:
```

initialize right grid points (excluding the two ends)

```
> for j from 1 to n-1 do uu[n,j] := 1-(a+j*h)^2: end do:
```

set up the linear equations using the Laplace stencil, for the interior grid points, using the lexicographical ordering

```
> eqs := []:
  for i from 1 to n-1 do
  for j from 1 to n-1 do
       eqs := [op(eqs),uu[i+1,j]+uu[i-1,j]+uu[i,j+1]+uu[i,j-1]-4*uu
  [i,j]=0]:
  end do;
  end do;
```

```
> eqs:
  nops(eqs);
> vars := [seq(seq(u[i,j],j=1..n-1),i=1..n-1)]:
  nops(vars);
```

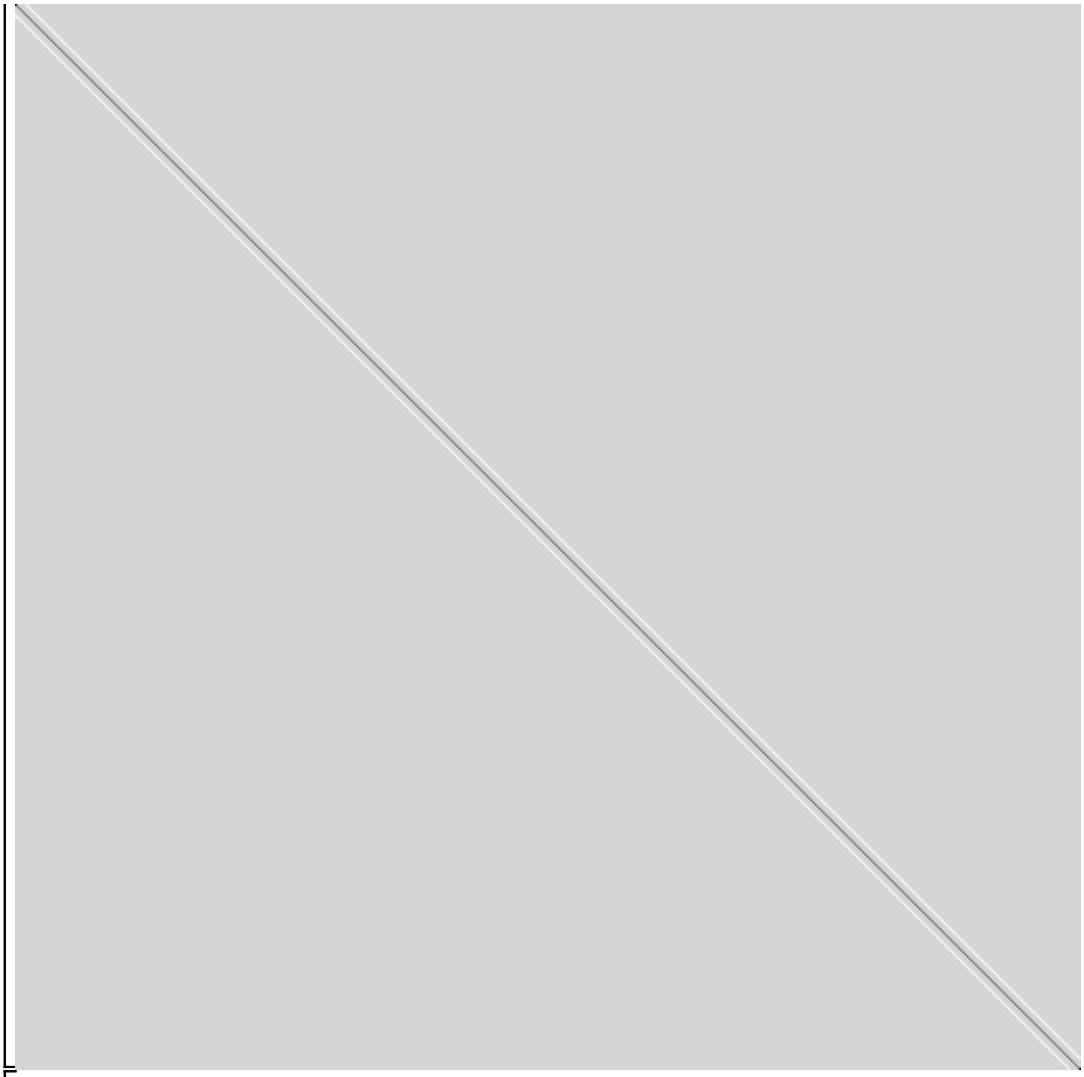
define the matrix and the constant vector for the above linear system of equations

```
> (A, b) := LinearAlgebra[GenerateMatrix]( eqs, vars ):
```

use Maple's Matrix Browser to visualize the structure of A (double click on the summary output below)

```
> A;
```

$$\begin{bmatrix} \textit{9801 x 9801 Matrix} \\ \textit{Data Type: anything} \\ \textit{Storage: rectangular} \\ \textit{Order: Fortran\_order} \end{bmatrix} \tag{9}$$



```
> approxSolutions:=LinearAlgebra[LinearSolve](A,b);
```

$$approxSolutions := \begin{bmatrix} 1 .. 9801\ Vector_{column} \\ Data\ Type: float_8 \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \quad (10)$$

```
> exact := x_e^2 - y_e^2;
```

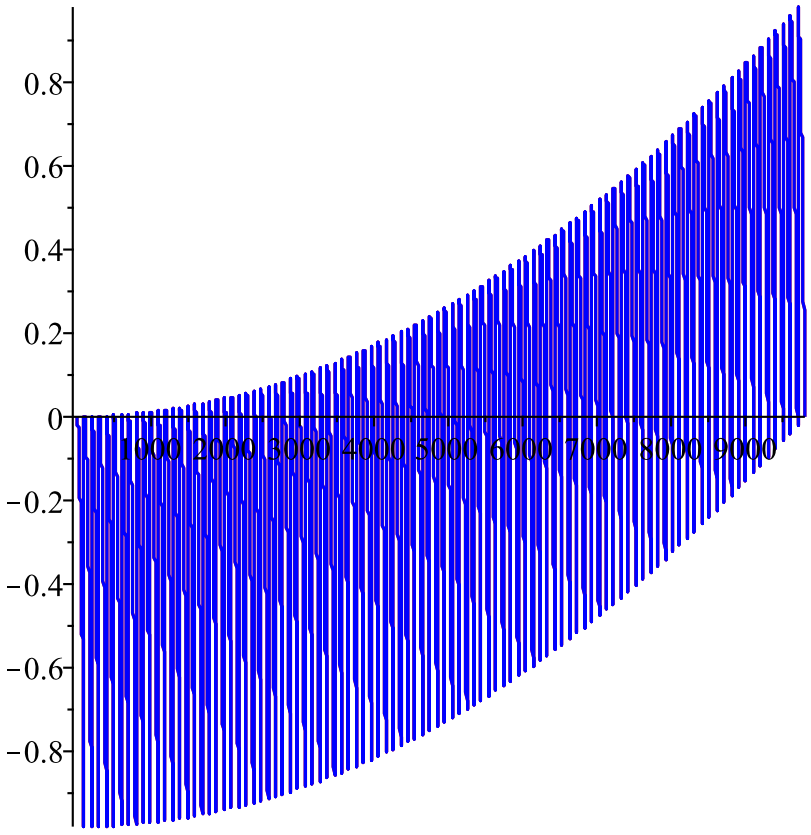
$$exact := x_e^{\,2} - y_e^{\,2} \quad (11)$$

```
> exactSolutions := [ ]:
```

```
for i from 1 to n − 1 do:
 for j from 1 to n − 1 do:
  exactSolutions := [op(exactSolutions), evalf(eval(exact, [x_e = a + i·h, y_e = a + j·h]))]
end do:
end do:
m := numelems(exactSolutions) :
```

```
> dataPointsApprox := plot([seq([i, evalf(approxSolutions[i])], i = 1..m)], style = line, color
    = red) : dataPointsExact := plot([seq([i, exactSolutions[i]], i = 1..m)], style = line, color
    = blue) :

  plots[display]({dataPointsApprox, dataPointsExact})
```



The above graph displays the exact solution in blue and approximate solutions in red. If no red is seen then the approximation give almost the same result as the exact solution. The x-axis simply represents the sequence of points and y-axis represents the actual values.