Scientific Computing II, SS 2012
Drawing Circles with ODEs

(worksheet inspired by the resp. example by G. Strang (Computational Science and Engineering)

```
> restart;
> with(DEtools):
> with(LinearAlgebra):
> with(plots):
>
```

A 2nd Order ODE

u the position vector; the second derivative of u, i.e., acceleration, is always directed towards the origin:

```
> circ_ode := {  diff( u[1](t),t,t ) = -u[1](t),
                 diff( u[2](t),t,t ) = -u[2](t) };
```

$$circ\_ode := \{\frac{d^2}{dt^2}u_1(t) = -u_1(t), \frac{d^2}{dt^2}u_2(t) = -u_2(t)\}$$

```
> dsolve( { op(circ_ode), u[1](0)=1, u[2](0)=0 }, {u[1](t),
  u[2](t)});
```

$$\{u_1(t) = \_C3\sin(t) + \cos(t), u_2(t) = \_C1\sin(t)\}$$

```
> usol := dsolve( { op(circ_ode),
            u[1](0)=1, u[2](0)=0,
            D(u[1])(0) = 0, D(u[2])(0) = 1 },
          {u[1](t), u[2](t)});
```

$$usol := \{u_1(t) = \cos(t), u_2(t) = \sin(t)\}$$

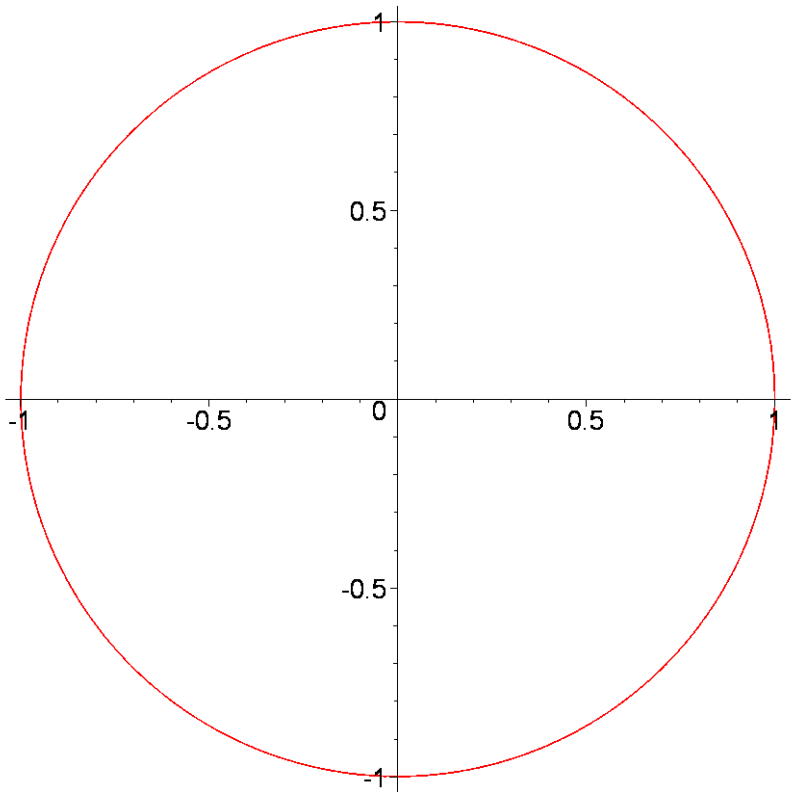
```
> us1 := subs(usol, u[1](t));
  us2 := subs(usol, u[2](t));
```

$$us1 := \cos(t)$$

$$us2 := \sin(t)$$

We obtain a circle as solution

```
> plot( [ us1,us2, t=0..100], scaling=CONSTRAINED );
```



Formulation as a 1st order ODE:

u: position vector, v: velocity vector

```
> circ_ode := {  diff( u[1](t),t ) = v[1](t),
                 diff( u[2](t),t ) = v[2](t),
                 diff( v[1](t),t ) = -u[1](t),
                 diff( v[2](t),t ) = -u[2](t) };
```

$$circ\_ode := \{\frac{d}{dt}u_1(t) = v_1(t), \frac{d}{dt}u_2(t) = v_2(t), \frac{d}{dt}v_1(t) = -u_1(t), \frac{d}{dt}v_2(t) = -u_2(t)\}$$

```
> uv_sol :=
  dsolve( { op(circ_ode),
            u[1](0)=1, u[2](0)=0, v[1](0)=0, v[2](0)=1 },
          { u[1](t), u[2](t), v[1](t), v[2](t)} );
```

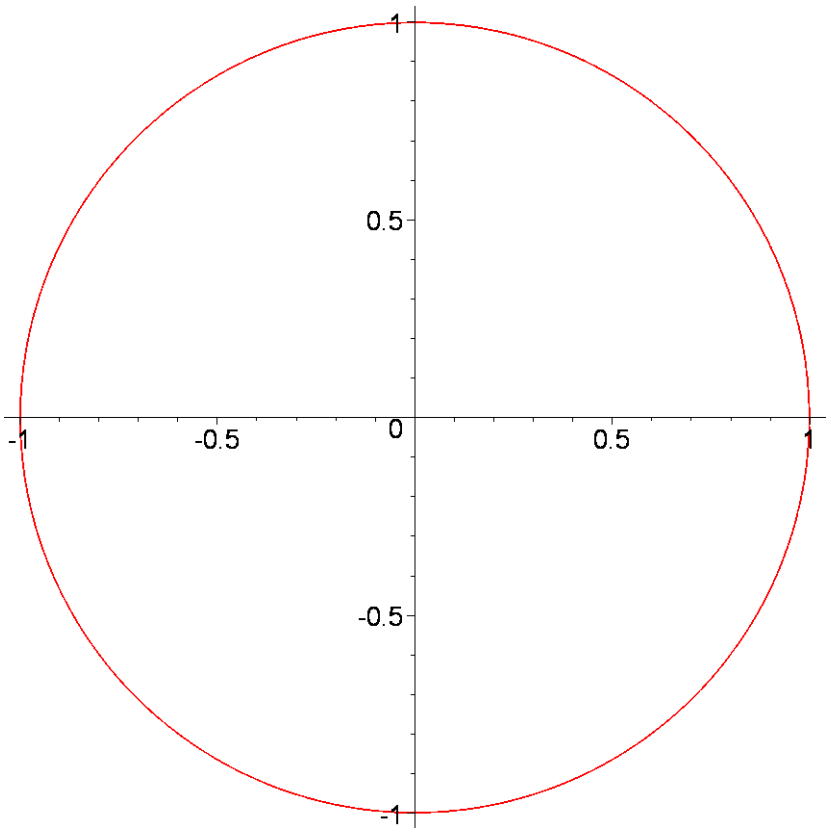
$$uv\_sol := \{u_1(t) = \cos(t), u_2(t) = \sin(t), v_1(t) = -\sin(t), v_2(t) = \cos(t)\}$$

```
> us1 := subs(uv_sol, u[1](t));
  us2 := subs(uv_sol, u[2](t));
```

$$us1 := \cos(t)$$

$$us2 := \sin(t)$$

```
> plot( [ us1,us2, t=0..100], scaling=CONSTRAINED );
```



Numerical Solution: Explicit Euler Scheme

We try an explicit Euler scheme to solve the PDE:
n is the number of time steps; tau the time step size;
-> we want to do m runs around the circle

```
> n := 500; m := 3;
  tau := evalf(m*2*Pi/n);
```

$$n := 500$$

$$m := 3$$

$$\tau := 0.03769911185$$

Define two arrays unum and vnum for the approximate solutions:

```
> u1num := Vector(1..n+1);
  u2num := Vector(1..n+1);
  v1num := Vector(1..n+1);
  v2num := Vector(1..n+1);
```

$$u1num := \begin{bmatrix} \text{501 Element Column Vector} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

$$u2num := \begin{bmatrix} \text{501 Element Column Vector} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

$$v1num := \left[\begin{array}{l}\text{501 Element Column Vector}\\ \text{Data Type: anything}\\ \text{Storage: rectangular}\\ \text{Order: Fortran\_order}\end{array}\right]$$

$$v2num := \left[\begin{array}{l}\text{501 Element Column Vector}\\ \text{Data Type: anything}\\ \text{Storage: rectangular}\\ \text{Order: Fortran\_order}\end{array}\right]$$

Initial conditions:

```
> u1num[1] := 1; u2num[1] := 0;
  v1num[1] := 0; v2num[1] := 1;
```

$$u1num_1 := 1$$

$$u2num_1 := 0$$

$$v1num_1 := 0$$

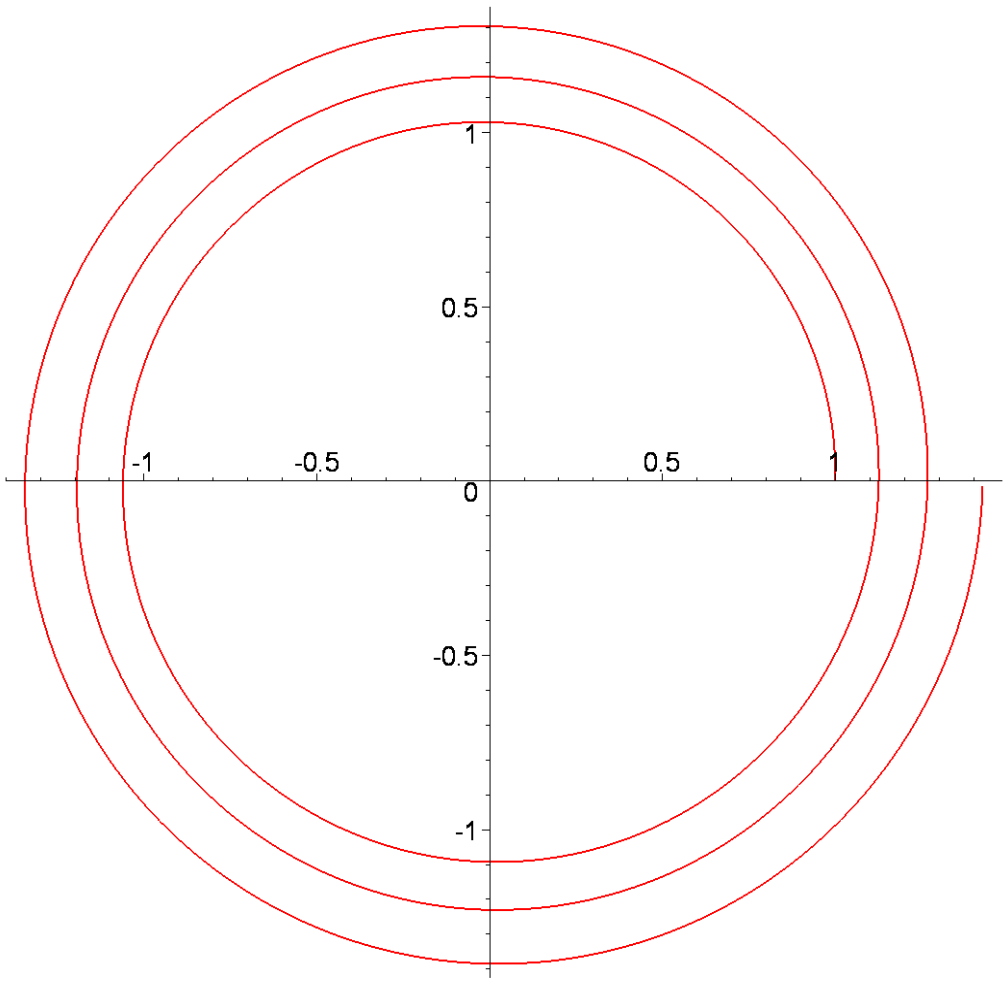
$$v2num_1 := 1$$

Solve system of ODEs via explicit Euler:

```
> for k from 1 to n do
      u1num[k+1] := u1num[k] + tau*(v1num[k]):
      u2num[k+1] := u2num[k] + tau*(v2num[k]):
      v1num[k+1] := v1num[k] + tau*(-u1num[k]):
      v2num[k+1] := v2num[k] + tau*(-u2num[k]):
  end do:
```

and plot the result:

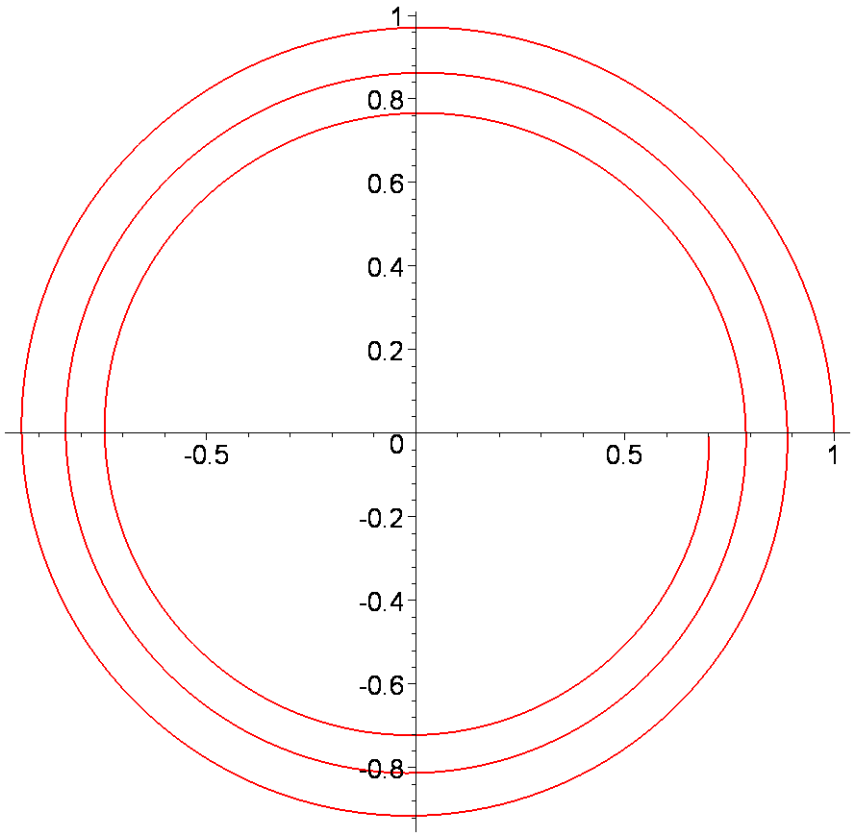
```
> plot( [ seq( [u1num[k], u2num[k]], k=1..n+1) ], thickness=2,
  scaling=CONSTRAINED );
```




```
>
```

Implicit Euler

```
> for k from 1 to n do
     nextstep := solve({
      u1new = u1num[k] + tau*(v1new),
      u2new = u2num[k] + tau*(v2new),
      v1new = v1num[k] + tau*(-u1new),
      v2new = v2num[k] + tau*(-u2new) }, {u1new, u2new, v1new,
  v2new}):
     u1num[k+1] := subs(nextstep, u1new);
     u2num[k+1] := subs(nextstep, u2new);
     v1num[k+1] := subs(nextstep, v1new);
     v2num[k+1] := subs(nextstep, v2new);
  end do:
> plot( [ seq( [u1num[k], u2num[k]], k=1..n+1) ], thickness=2,
  scaling=CONSTRAINED );
```




```
>
```

Trapezoid Rule (Crank-Nicholson, implicit method)

```
> n := 100; m := 3;
  tau := evalf(m*2*Pi/n);
  u1num := Vector(1..n+1): u2num := Vector(1..n+1):
  v1num := Vector(1..n+1): v2num := Vector(1..n+1):
```

$$n := 100$$

$$m := 3$$

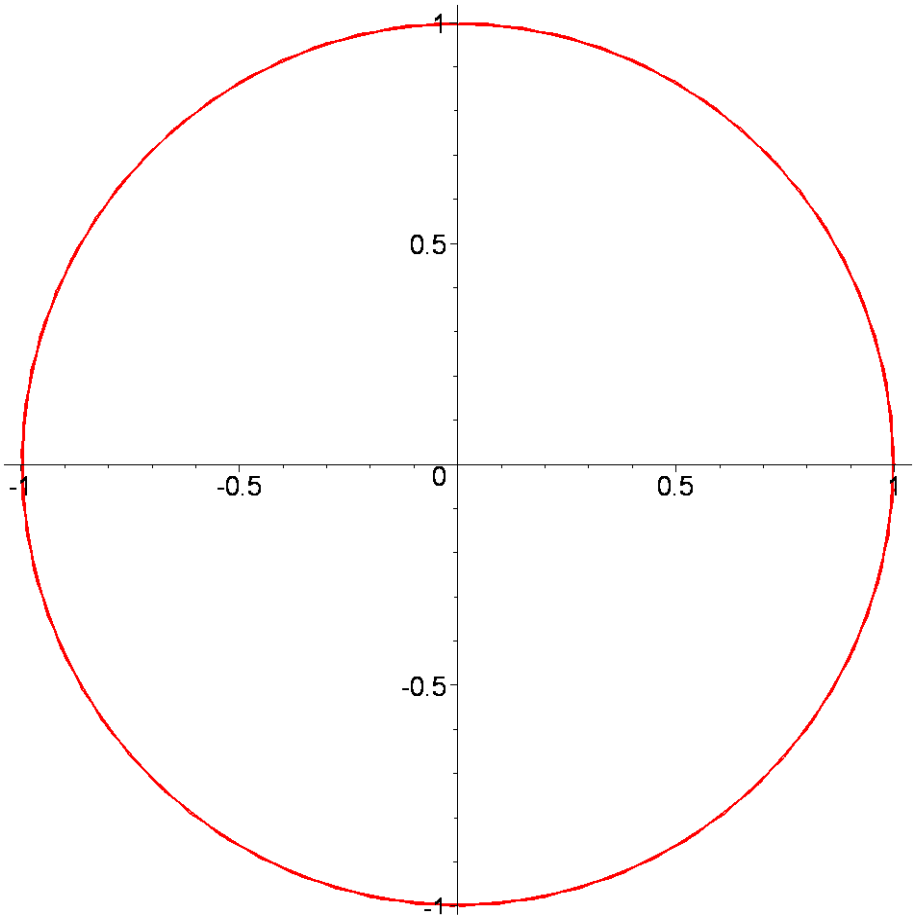
$$\tau := 0.1884955592$$

```
> u1num[1] := 1: u2num[1] := 0:
  v1num[1] := 0: v2num[1] := 1:
> for k from 1 to n do
     nextstep := solve({
      u1new = u1num[k] + tau*(v1num[k]+v1new)/2,
      u2new = u2num[k] + tau*(v2num[k]+v2new)/2,
      v1new = v1num[k] + tau*(-u1num[k]-u1new)/2,
      v2new = v2num[k] + tau*(-u2num[k]-u2new)/2 },
      {u1new, u2new, v1new, v2new}):
     u1num[k+1] := subs(nextstep, u1new);
     u2num[k+1] := subs(nextstep, u2new);
     v1num[k+1] := subs(nextstep, v1new);
     v2num[k+1] := subs(nextstep, v2new);
  end do:
```

The implicit Crank-Nicholson scheme turns out to work fine;

however, as an implicit method, we will not be able to use it for molecular dynamics (it's simply too expensive)

```
> plot( [ seq( [u1num[k], u2num[k]], k=1..n+1) ], thickness=2,
  scaling=CONSTRAINED );
```




```
>
```

An alternative (simple) Leapfrog Scheme

```
> n := 100; m := 3;
  tau := evalf(m*2*Pi/n);
  u1num := Vector(1..n+1): u2num := Vector(1..n+1):
  v1num := Vector(1..n+1): v2num := Vector(1..n+1):
```

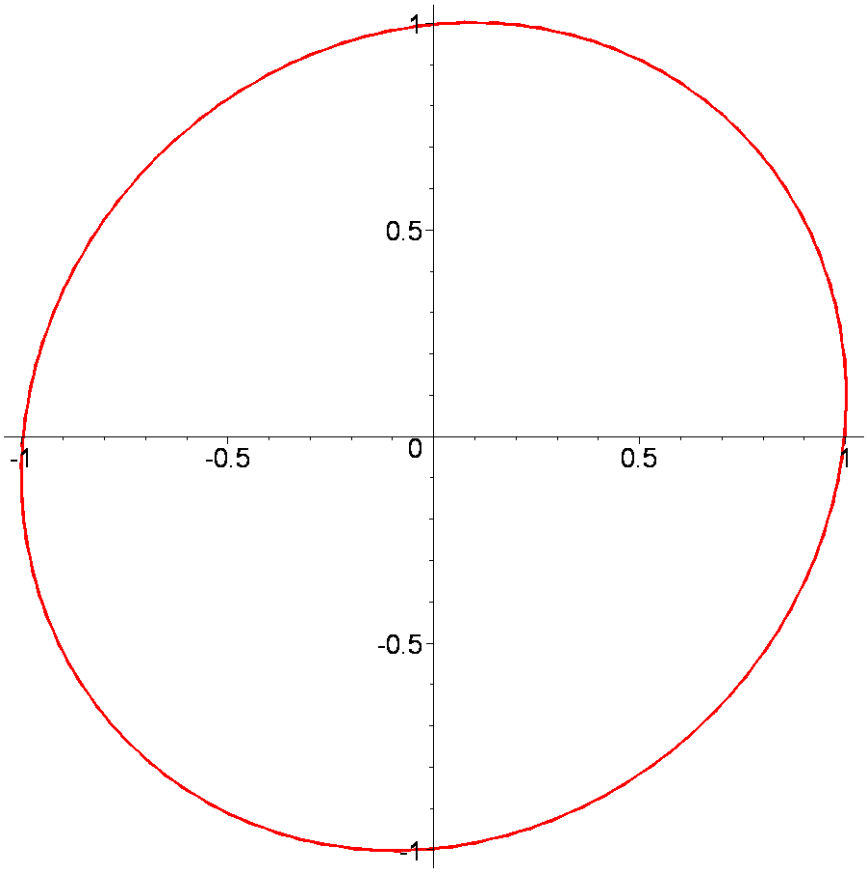
$$n := 100$$

$$m := 3$$

$$\tau := 0.1884955592$$

```
> u1num[1] := 1: u2num[1] := 0:
  v1num[1] := 0: v2num[1] := 1:
> for k from 1 to n do
     u1num[k+1] := u1num[k] + tau*(v1num[k]):
     u2num[k+1] := u2num[k] + tau*(v2num[k]):
     v1num[k+1] := v1num[k] + tau*(-u1num[k+1]):
     v2num[k+1] := v2num[k] + tau*(-u2num[k+1]):
  end do:
> plot( [ seq( [u1num[k], u2num[k]], k=1..n+1) ], thickness=2,
```

```
scaling=CONSTRAINED );
```




```
>
```

Velocity Verlet

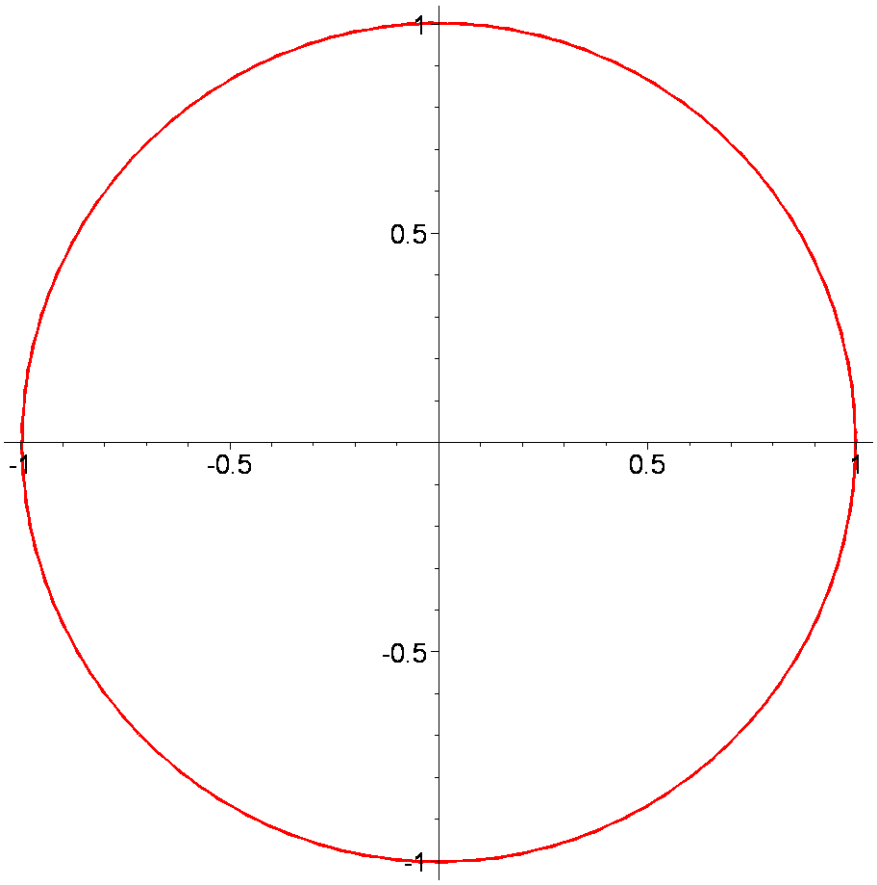
```
> n := 100; m := 3;
  tau := evalf(m*2*Pi/n);
  u1num := Vector(1..n+1): u2num := Vector(1..n+1):
  v1num := Vector(1..n+1): v2num := Vector(1..n+1):
```

$$n := 100$$

$$m := 3$$

$$\tau := 0.1884955592$$

```
> u1num[1] := 1: u2num[1] := 0:
  v1num[1] := 0: v2num[1] := 1:
> for k from 1 to n do
     v1ver := v1num[k] + tau*(-u1num[k])/2:
     v2ver := v2num[k] + tau*(-u2num[k])/2:
     u1num[k+1] := u1num[k] + tau*(v1ver):
     u2num[k+1] := u2num[k] + tau*(v2ver):
     v1num[k+1] := v1ver + tau*(-u1num[k+1])/2:
     v2num[k+1] := v2ver + tau*(-u2num[k+1])/2:
  end do:
> plot( [ seq( [u1num[k], u2num[k]], k=1..n+1) ], thickness=2,
  scaling=CONSTRAINED );
```



[v