

2 Numerical methods for ODE's

2.1 A very high order method

Use the operator identity

$$\frac{d}{dt} = \frac{2}{k} \operatorname{arcsinh} \frac{\delta}{2}$$

with

$$\delta = E^{1/2} - E^{-1/2} \tag{1}$$

$$\delta q(t) = q(t + k/2) - q(t - k/2) \tag{2}$$

to derive an $O(k^8)$ formula for integrating

$$\begin{aligned} q' &= f(t, q) = \sin(t + q) \\ q(t = 0) &= 1 \end{aligned}$$

from $t = 0$ to $t = 1$. Note that nothing has been specified about the time at which you should evaluate f . Try out various choices. Apply the formula with various step sizes $k = 2^{-p}$, $p = 1, 2, \dots, 10$. Carry out a convergence analysis, i.e. compute the relative error at $t = 1$ and draw a $\lg - \lg$ plot of the relative error versus step size. Analyze the plot. Is the convergence as you would expect from your derivation of the formula?

Solution Using the `s1centered` function from the `lab2.nb` Mathematica notebook we find the desired $O(k^8)$ approximation:

$$\frac{dq}{dt}(t^n) = \frac{1}{k} \sum_{j=1}^4 c_j (Q^{n+j-1/2} - Q^{n-j+1/2}) + O(k^8)$$

with the coefficients

$$\begin{array}{cccc} j & 1 & 2 & 3 & 4 \\ c_j & \frac{1125}{1024} & -\frac{245}{3072} & \frac{49}{5120} & -\frac{5}{7168} \end{array} .$$

Note the symmetries in the finite difference approximation and the compact rendition of the formula. Applying the above approximation to the ODE for this problem leads to

$$\sum_{j=1}^4 c_j (Q^{n+j-1/2} - Q^{n-j+1/2}) = k \sin(t^n + Q^n) ,$$

a formula that would be applied starting with $n = 7/2$ so that the smallest index that arises corresponds to $Q^0 = 1$, the initial value we are given. The f function is evaluated at t^n corresponding to the point at which we've approximated the derivative. Other choices would lead to an imprecise method as can be verified by a series expansion. In a computer program we would implement the above formula as

$$Q^{n+7/2} = Q^{n-7/2} - \sum_{j=1}^3 a_j (Q^{n+j-1/2} - Q^{n-j+1/2}) + k \sin(t^n + Q^n), \quad n = 7/2, 4, 9/2, \dots$$

with $a_j = c_j/c_4$. Notice the way the computation is organized: we take differences of the symmetric values around Q^n and then multiply by a_j and add the contributions. This tends to aid in maintaining computer precision (see Bonus question from HW 1). The half-indices are inconvenient for computer storage so we will use

$$q[2n + 1] = Q^n$$

with $q[n]$ denoting the computer code vector where we'll store the $Q^{(n-1)/2}$ value. An actual computer-language implementation will look like

$$q[1] = Q^0$$

$$q[2(n + 4)] = q[2(n - 3)] - \sum_{j=1}^3 a_j (q[2(n + j)] - q[2(n - j + 1)]) + k \sin(t^n + q[2n + 1]),$$

for $n = 7/2, 4, 9/2, \dots, N - 7/2$

We are faced with the problem of generating the large number of startup values that the algorithm needs Q^l for $l = \frac{1}{2}, 1, \dots, \frac{13}{2}$. This must be done in a way that maintains the desired $O(k^8)$ accuracy. The truncation error for this algorithm (from the Mathematica series expansion) is

$$\tau^n = \frac{35q^{(9)}(\xi^n)k^8}{294912}.$$

The one-step error is one order higher

$$\omega^n = \frac{35q^{(9)}(\xi^n)k^9}{294912}.$$

There are various strategies one can adopt in order to generate these values:

1. We could use an exact, analytical solution to evaluate the starting values. Of course this defeats the purpose of the whole exercise. We're interested in a numerical algorithm to find the solution because the analytical route is impossible or too complicated. We're free to use analytical values as an initial verification, but this does not eliminate the need to devise procedures that use just the information at hand, namely $q(0) = 1$ and $f(t, q) = \sin(t + q)$.

2. We could apply an $O(k^7)$ single step method. Note that we can use an algorithm one order of precision less since we'll only be applying it a small number of times as opposed to the main algorithm which is applied $O(1/k)$ times (that's why the truncation order of precision is one less than the one-step error). We could generate such a method through the Taylor series method or look up an $O(k^7)$ Runge-Kutta method (e.g. Abramowitz & Stegun, Handbook of Mathematical Functions).
3. We could write an $O(k^7)$ multiple-step method. One difficulty that arises with this approach is that we would have to generate start-up values for this algorithm through an $O(k^6)$ method presumably, and this would lead to a cascade of sub-problems - possible in principle but a bit too much work.
4. We could apply a lower order method with a smaller time step. The main, $O(k^8)$ method has the property that if we halve the step size the error goes down by a factor of $1/2^8 = 1/256$. Consider the common $O(k^4)$ Runge-Kutta method (RK4)

$$K_1 = k f(t^n, Q^n) \tag{3}$$

$$K_2 = k f(t^n + k/2, Q^n + K_1/2) \tag{4}$$

$$K_3 = k f(t^n + k/2, Q^n + K_2/2) \tag{5}$$

$$K_4 = k f(t^n + k, Q^n + K_3) \tag{6}$$

$$Q^{n+1} = Q^n + \frac{1}{6} (K_1 + 2K_2 + 2K_3 + K_4) \tag{7}$$

This method has a truncation error of $O(k^4)$

$$\tau_{RK}^n = Cq^{(5)}(\xi_{RK}^n)k^4$$

and leads to a reduction of the error by a factor of $1/2^4 = 1/16$ for every halving of the step size. If we wish to avoid the difficulty of the first two approaches we can simply apply the above algorithm with a smaller step size. For any given step size k of the $O(k^8)$ method we would use RK4 with a step size k_1 given by the condition of obtaining comparable truncation errors

$$\left| Cq^{(5)}(\xi_{RK}^n) \right| k_1^4 = \frac{35q^{(9)}(\xi^n)k^8}{294912} .$$

We are faced of course with the problem of establishing values for ξ_{RK}^n , ξ^n ; this is not as bad as it first seems if we think about it a bit. If q has bounded derivatives up to 9th order we obtain simply that

$$k_1 = Ak^2$$

with A some unknown constant. Note that the above formula states that for every halving of k the RK4 step size should be reduced to $1/4$

of that previously used. The constant A affects the overall error for a given step size but not the convergence behavior, i.e. how the error decreases with decreasing step size. We could be more precise and look up what C is for RK4 and do rigorous bounds on the derivatives of q , but this won't really affect the behavior of the $O(k^8)$ method we're looking at as long as the derivatives of q are indeed bounded, so we can simply choose $A = 1$ for our tests. (Note: This might seem to be the proverbial rabbit-out-of-the-hat and perhaps unfair as a homework question since such an approach was not discussed in class. I disagree: the above method is just a simple verification of understanding of what we mean by truncation error - an indication of the expected convergence behavior.)

However, before we actually use one of the above approaches to generate start-up values, it is a good idea to test the algorithm on a simpler problem that we fully understand, e.g.

$$\begin{aligned} q' &= q \\ q(t=0) &= 1 \end{aligned}$$

with the exact solution

$$q(t) = e^t.$$

For this problem we can use the exact solution to give exact startup values. A Matlab program to carry out this computation is given below along with a graph of the convergence behavior. Please look carefully at the implementation: the notation from the theory is followed as closely as possible, comments describing the code are included, parameters are given variable names so they can be easily changed. Follow these good coding practices. It is readily apparent that the algorithm is **not** converging. If the algorithm does not converge on this simple, test problem there is no point to trying it out on a more complicated problem. Any further computations would be a waste of time; we must first establish whether the algorithm exhibits bad numerical properties.

Comment on grading. So what am I looking for when I'm grading this homework question? The main concern is whether you are systematically investigating as many aspects of the algorithm as feasible and whether it is clear that there is an orderly progression of subquestions that you pose and then successfully answer. This is the essence of the scientific research process. Specifically:

1. Deriving the method is immediate since the tools have already been provided (**lab2.nb**) so I'm looking for a presentation of the method that makes it clear that you're checking the derivation and making it easy for me to check it also (imagine I'm the reviewer on one of the papers you've submitted to a journal - it is **your** responsibility to

ensure a clear and concise presentation, otherwise the reviewer gets annoyed and rejects the paper).

2. After derivation of the method and putting it in algorithmic form (i.e. the way it will be implemented in a computer program) I'm looking to see if you're assessing the merits and drawbacks of the method, e.g.:
 - explicit or implicit?
 - stencil width? computational complexity?
 - is the information in the initial problem sufficient to make the algorithm work? (in this case no, we need starting values)
3. I am then looking to see if you devise simple tests to check whether the work you've carried out is correct. This necessary whenever we look at a new algorithm.
4. Many methods may seem overly complex at first glance. Whenever a specific method is outside of the realm of previous experience, the profitable way forward is to imagine simpler variants and gain experience by analyzing those. For this problem, if the $O(k^8)$ method seemed too complicated, you could have tried to answer the problem questions on an $O(k^2)$ method and then on an $O(k^4)$ method. Had these been analyzed correctly according to the guidelines set above the full homework point would be awarded.
5. I expect you to recognize unreasonable requests from how the question is posed. This is called "critical reading ability" or more simply the ability to identify some desired goal as impossible or unrealistic.
6. Though due consideration will be given to non-native English speakers, it is expected that homeworks will be drafted using complete and clear sentences.
7. Final observation: it is really very easy for an instructor to determine whether a homework is being done as a learning process or as a simple chore to be carried out as quickly as possible. Even if you do not completely solve a problem, the homework point will be awarded in full if the learning and investigation process is readily apparent. Even if you present a correct problem solution, you will **not** be awarded the homework point if the methods and reasoning by which you arrived at that solution are not presented, are inappropriate or materially incomplete. Also note that the time spent on a question need not be an indication of the quality of a solution. One can spend inordinate amounts of time on solving a problem if the requisite theory is poorly understood or if the investigative process is disorderly.

```
T=1; % final time
q0=1; % initial condition
Nj=3; % number of terms in algorithm sum
```

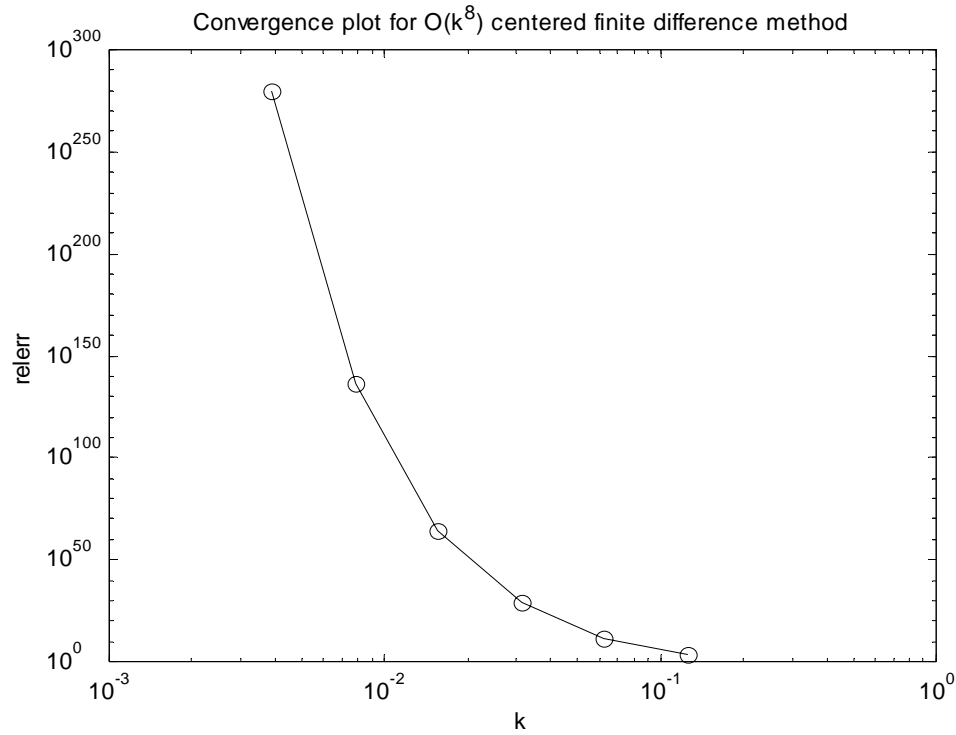
```

r=14; % number of algorithm steps
% finite difference formula coefficients
c=[1125/1024 -245/3072 49/5120 -5/7168];
% algorithm coefficients
a=c(1:3)/c(4);
np=10;
relerr=ones(np,1); kp=ones(np,1);
% exact solution for tests of algorithm
exactq=inline('exp(t)', 't'); f=inline('exp(t)', 't');
for p=1:np
    k=2^(-p); % step-size
    kp(p)=k;
    N=T/k;
    q=zeros(2*N+1,1); % number of memory locations we'll need
    q(1)=q0; % set initial condition
    % Set starting values
    for j=2:r
        t=(j-1)*k/2;
        q(j)=feval(exactq,t);
    end
    % Apply O(k^8) method
    for n=7/2:1/2:N-7/2
        t=n*k;
        m=2*(n+4);
        q(m)=q(2*(n-3));
        for j=1:Nj
            q(m) = q(m) - a(j)*(q(2*(n+j)) - q(2*(n-j+1)));
        end
        fn=feval(f,t);
        q(m) = q(m) + k*fn;
    end
    % compute relative error
    exactq1=feval(exactq,T);
    relerr(p)=abs((exactq1-q(2*N+1))/exactq1);
end
% plot the errors
loglog(kp,relerr,'-ok');
title('Convergence plot for O(k^8) centered finite difference method')
xlabel('k');
ylabel('rel err');

```

2.2 But is it stable?

Verify the zero-convergence of the above method for various choices of when to evaluate $f(t, q)$ such as at the least time that appears in the approximation of



q' , the greatest time or the average of these two.

Solution This is a linear multistep method. We know that we can establish the convergence in the limit of step size going to zero, $k \rightarrow 0$ by looking at the simple model problem

$$\begin{aligned} q' &= \lambda q \\ q(t=0) &= q_0 \end{aligned}$$

(the argument is that λ plays the role of a local Lipschitz constant). For such a problem our method would be

$$\sum_{j=1}^4 c_j (Q^{n+j-1/2} - Q^{n-j+1/2}) = \lambda k Q^n$$

with the rhs evaluated at t^n the midpoint between the extremal time levels encountered in the method. We would like to avoid half-value indices so we'll apply the method for a $2k$ step size

$$\sum_{j=1}^4 c_j (Q^{n+2j-1} - Q^{n-2j+1}) = 2\lambda k Q^n .$$

The convergence behavior in the limit of $k \rightarrow 0$ can mean one of two things: (1) convergence in exact arithmetic; (2) convergence in approximate arithmetic. To establish the convergence in exact arithmetic we must investigate the limit

$$\lim_{k \rightarrow 0} E^n$$

with $E^n = Q^n - q(t^n)$ the error in the numerical solution and $kn = t^n$ some fixed, finite time. The form of the algorithm suggests that we construct the expression

$$\sum_{j=1}^4 c_j (q(t^{n+2j-1}) - q(t^{n-2j+1}))$$

the Taylor series expansion of which is

$$\sum_{j=1}^4 c_j (q(t^{n+2j-1}) - q(t^{n-2j+1})) = 2kq'(t^n) - \frac{35q^{(9)}(\xi^n)(2k)^9}{294912} = 2kq'(t^n) - \omega^n$$

For our model problem, $q'(t^n) = \lambda q(t^n)$. Subtracting this from the $O(k^8)$ method we're studying leads to the error recursion relation

$$\sum_{j=1}^4 c_j (E^{n+2j-1} - E^{n-2j+1}) = 2\lambda k E^n - \omega^n$$

The $k = 0$ limit would correspond to

$$\sum_{j=1}^4 c_j (E^{n+2j-1} - E^{n-2j+1}) = 0$$

and we would expect this relation to predict that the error is zero. As it stands, we must make additional hypothesis about the algorithm to proceed. We need starting values. Let's first assume that these are given exactly such that $E^0 = \dots = E^{13} = 0$. Then by the above relation $E^{14} = 0$ and all successive values are also zero so the algorithm would be convergent in the limit $k \rightarrow 0$ with exact arithmetic and exact starting values. Now assume that the starting error values are not zero. We can solve for the roots of the characteristic polynomial of the above recurrence relation

$$c_1 \left(\zeta - \frac{1}{\zeta} \right) + c_2 \left(\zeta^3 - \frac{1}{\zeta^3} \right) + c_3 \left(\zeta^5 - \frac{1}{\zeta^5} \right) + c_4 \left(\zeta^7 - \frac{1}{\zeta^7} \right) = 0$$

using a number of mathematical packages (Mathematica, Maple, Matlab). The roots $\zeta_{1,2} = \pm 1$ are immediately apparent. The method would lead to growth of initial errors if any of the roots are greater than 1 in absolute value. Given our previous numerical experiments we suspect this is the case and indeed it is easy to find a root such as $\zeta = 3.71406$. We conclude that the algorithm would lead to amplification of any initial errors

we make in the starting values even in exact arithmetic and even more so in approximate arithmetic so the only case where the algorithm is zero-convergent is for exact starting values and exact arithmetic. Note that exact arithmetic is attainable if we work with rational number approximations of the reals and the particular ODE leads to function giving rational values for rational arguments (nice research project: devise such an ODE and use rational arithmetic to see that the method does indeed converge as you decrease k !).

Choosing different rhs evaluation points (e.g. t^{n+7} or t^{n-7} as suggested in the problem) would lead to

$$\begin{aligned} \sum_{j=1}^4 c_j (E^{n+2j-1} - E^{n-2j+1}) &= 2\lambda k (Q^{n\pm 7} - q(t^n)) - \omega^n \\ &= 2\lambda k (Q^{n\pm 7} - q(t^{n\pm 7}) + q(t^{n\pm 7}) - q(t^n)) - \omega^n \\ &= 2\lambda k E^{n\pm 7} + q'(t^n)(\pm 7k) + q''(t^n)(\pm 7k)^2/2 + \dots - \omega^n \end{aligned}$$

Interestingly enough this still gives a rhs equal to zero in $k = 0$ limit so in exact arithmetic, exact starting values we would eventually obtain the correct solution as the step size is decreased to zero. The reason behind this surprising result is that we're evaluating f at a finite number of steps away from t^n and as $k \rightarrow 0$ so also does $pk \rightarrow 0$ for any finite p . For any error in the startup values or from inexact arithmetic the error would grow without bound since we still have the same lhs characteristic polynomial.

2.3 Is it stable for practical computations?

Now verify the absolute stability of the above algorithms. Use the boundary locus method and comment on what you obtain. Reinterpret your results from the first problem in light of what you learn from the absolute stability analysis.

Solution The characteristic polynomial of the lhs is

$$\rho(\zeta) = \left[c_1 \left(\zeta - \frac{1}{\zeta} \right) + c_2 \left(\zeta^3 - \frac{1}{\zeta^3} \right) + c_3 \left(\zeta^5 - \frac{1}{\zeta^5} \right) + c_4 \left(\zeta^7 - \frac{1}{\zeta^7} \right) \right] \zeta^7$$

and that of the rhs is

$$\sigma(\zeta) = 2\zeta^7$$

when we evaluate f at t^n so the boundary locus when $\zeta = e^{i\theta}$ is given by

$$\begin{aligned} z(\theta) &= \frac{1}{2} [c_1 (e^{i\theta} - e^{-i\theta}) + c_2 (e^{3i\theta} - e^{-3i\theta}) + c_3 (e^{5i\theta} - e^{-5i\theta}) + c_4 (e^{7i\theta} - e^{-7i\theta})] \\ &= i [c_1 \sin \theta + c_2 \sin 3\theta + c_3 \sin 5\theta + c_4 \sin 7\theta] \end{aligned}$$

As θ goes from 0 to 2π , $z(\theta)$ describes a segment on the imaginary axis. A point inside of this region is $z = 0$ which leads

$$\pi(\zeta; z) = \rho(\zeta)$$

which we've seen has at least one root ($\zeta = 3.71406$) greater than 1 in absolute value so the method is not absolutely stable in this region (or indeed zero-stable given our choice of $z = 0$). Outside the slit, say at $z = -1$ we obtain the polynomial

$$\pi(\zeta; z) = \rho(\zeta) + \sigma(\zeta)$$

for which again we find a root of absolute value greater than 1, e.g. $\zeta = 3.97563$.

If we evaluate f at $t^{n\pm 7}$ we have the same $\rho(\zeta)$ but

$$\sigma(\zeta) = 2\zeta^{7\pm 7},$$

so the locus is given by

$$\begin{aligned} z(\theta) &= i [c_1 \sin \theta + c_2 \sin 3\theta + c_3 \sin 5\theta + c_4 \sin 7\theta] e^{\mp 7i\theta} \\ &= [c_1 \sin \theta + c_2 \sin 3\theta + c_3 \sin 5\theta + c_4 \sin 7\theta] [i \cos(7\theta) \mp \sin(7\theta)] \end{aligned}$$

The locii are rendered below. We obtain ellipses in the complex z -plane. For the t^{n-7} locus we can choose $z = -1/2$ and find a root $\zeta = 3.71408$ of $\pi(\zeta; -1/2) = \rho(\zeta) + 1$ showing the inside of the ellipse is not a region of absolute stability. For $z = 1/2$ we find $\zeta = 3.71404$ of $\pi(\zeta; -1/2) = \rho(\zeta) - 1$ showing that the exterior of the ellipse is also not a region of absolute stability so the method is unconditionally absolutely unstable. For the t^{n+7} locus we look at $z = -1/2$ and ask for the roots of $\pi(\zeta; -1/2) = \rho(\zeta) + \zeta^{14}$ and find one greater than 1 in absolute value and also for $z = 1/2$ (see Mathematica command below).

In all cases we find that the algorithm is absolutely unstable showing that the rather natural idea of getting a higher order method through a more accurate central difference approximation **does not work**.

2.4 Extension to PDE's

Write down the algorithm obtained by using a second-order, centered finite difference approximation for the spatial derivative and the approximation of the time derivative from problem 1 for the problem

$$\begin{cases} q_t = q_{xx} \\ q(x, 0) = \sin(\pi x) \\ q(0, t) = 0, q(1, t) = 0 \end{cases} \quad (8)$$

on $(x, t) \in [0, 1] \times [0, 1]$. Draw the stencil of the resulting algorithm. Think of a way to start the computation. Check the stability. Try out a few computations with decreasing step sizes and comment on whether the results converge or not.

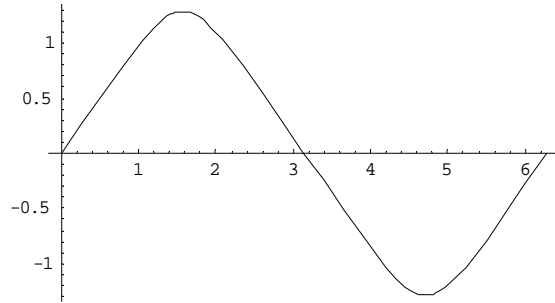


Figure 1: Slit along imaginary axis for f evaluated at t^n .

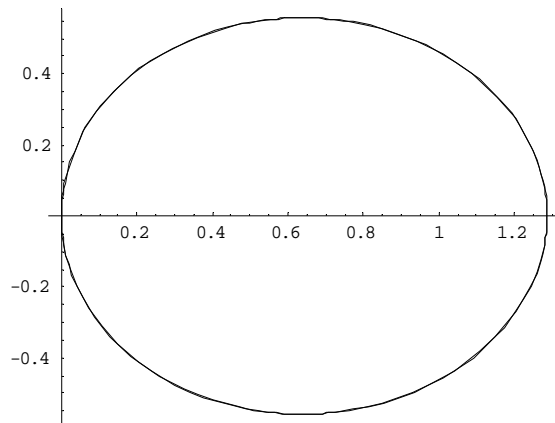


Figure 2: Boundary locus for f evaluated at t^{n+7}

Solution The approximation for the spatial derivative would lead to

$$q_{xx}(x_i, t^n) \cong \frac{Q_{i+1}(t^n) - 2Q_i(t^n) + Q_{i-1}(t^n)}{h^2}$$

with $x_i = ih$, $t^n = nk$ and h, k the spatial, temporal step sizes. The stencil would link 14 values along the time axis at x_i with Q_{i+1}^n, Q_{i-1}^n . This is a complicated stencil to implement. To start the computation we would again need to generate starting values, this time across all x_i values, for the first 13 time levels. We could do this by the RK4 procedure mentioned in Problem 1. Indeed we could try out a few computations, but they would be pointless - we know that the algorithm used for time discretization is absolutely unstable. (Remember: just because someone tells you to jump off a cliff is not sufficient reason to actually do it).

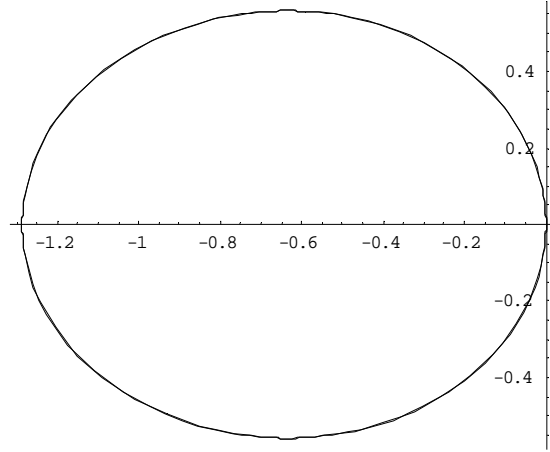


Figure 3: Boundary locus for f evaluated at t^{n-7}

In[170]:= $\rho[\xi] = \frac{1225}{1024} (\xi - 1/\xi) - \frac{245}{3072} (\xi^3 - 1/\xi^3) + \frac{49}{5120} (\xi^5 - 1/\xi^5) - \frac{5}{7168} (\xi^7 - 1/\xi^7)$

Out[170]= $\frac{1225 (-\frac{1}{\xi} + \xi)}{1024} - \frac{245 (-\frac{1}{\xi^3} + \xi^3)}{3072} + \frac{49 (-\frac{1}{\xi^5} + \xi^5)}{5120} - \frac{5 (-\frac{1}{\xi^7} + \xi^7)}{7168}$

In[194]:= $\text{NSolve}[\rho[\xi] \xi^7 - \xi^{14} == 0, \xi]$

Out[194]= $\{\{\xi \rightarrow -0.496815 + 1.0012 i\}, \{\xi \rightarrow -0.496815 - 1.0012 i\}, \{\xi \rightarrow 0.496815 + 1.0012 i\},$
 $\{\xi \rightarrow 0.496815 - 1.0012 i\}, \{\xi \rightarrow -0.88776 + 0.24316 i\}, \{\xi \rightarrow -0.88776 - 0.24316 i\}, \{\xi \rightarrow 0.88776 + 0.24316 i\},$
 $\{\xi \rightarrow 0.88776 - 0.24316 i\}, \{\xi \rightarrow -0.209484 + 0.220826 i\}, \{\xi \rightarrow -0.209484 - 0.220826 i\},$
 $\{\xi \rightarrow 0.209484 + 0.220826 i\}, \{\xi \rightarrow 0.209484 - 0.220826 i\}, \{\xi \rightarrow 0.269246\}, \{\xi \rightarrow -0.269246\}$

In[195]:= $\text{NSolve}[\rho[\xi] \xi^7 + \xi^{14} == 0, \xi]$

Out[195]= $\{\{\xi \rightarrow 0. + 1.14716 i\}, \{\xi \rightarrow 0. - 1.14716 i\}, \{\xi \rightarrow -0.820006 + 0.642334 i\},$
 $\{\xi \rightarrow -0.820006 - 0.642334 i\}, \{\xi \rightarrow 0.820006 + 0.642334 i\}, \{\xi \rightarrow 0.820006 - 0.642334 i\},$
 $\{\xi \rightarrow 0.850892\}, \{\xi \rightarrow -0.850892\}, \{\xi \rightarrow 0.209491 + 0.22083 i\}, \{\xi \rightarrow 0.209491 - 0.22083 i\},$
 $\{\xi \rightarrow -0.209491 + 0.22083 i\}, \{\xi \rightarrow -0.209491 - 0.22083 i\}, \{\xi \rightarrow -0.269248\}, \{\xi \rightarrow 0.269248\}$

2.5 Bonus - Computer generated algorithms

The formal series for deriving finite difference approximations

$$\frac{d}{dt} = \frac{1}{k} \left(\Delta_+ - \frac{1}{2} \Delta_+^2 + \frac{1}{3} \Delta_+^3 - \dots \right) = \frac{1}{k} \left(\Delta_- + \frac{1}{2} \Delta_-^2 + \frac{1}{3} \Delta_-^3 + \dots \right) \quad (9)$$

is well suited to computer implementation using a symbolic processing package (Mathematics, Maple). Try to write a program that carries out an $O(k^m)$ in time, $O(h^n)$ in space, finite difference approximation for the IVP from the previous problem for arbitrary m, n . Most symbolic packages have routines to directly output Fortran or C code; try to do this for your finite difference method. Also, it should be possible to carry out zero-stability and absolute stability analysis in the symbolic program.

Hint In Mathematica the relevant functions are `FortranForm`, `CForm`. Also check `TeXForm` for helping write impressive formulas in theses. Maple has an equivalent capability.