Problemas de contorno: Influencia de pequeños parámetros en ED.

Capítulo 5 – Libro **+** ***Hoja de problemas 10*** **+** ***Complementos***

Resumen de contenidos:

- **Diferencias finitas**
- **La función "*bvp4c*"**
- **Análisis espectral para un problema "stiff"**

% Este cuaderno tiene un carácter complementario. En la primera parte se usan las funciones Matlab ffinitge.m y bvp4c para aproximar soluciones de problemas de contorno planteados en la hoja de problemas número 8; estas aproximaciones se comparan con soluciones explícitas, si se pueden calcular, o con las aproximaciones obtenidas con métodos de tiro y de elementos finitos de las hojas 4 y 8 respectivamente (ver enlaces ***Hoja de problemas 4*** *y* ***Hoja de problemas 8****)*

% En la segunda parte, se introduce un problema espectral "stiff", y se ejecuta un programa (fvpstiff.m, con un método en diferencias finitas) que nos muestra los distintos modos propios de vibración asociados a las altas y bajas frecuencias. El análisis asintótico, que explica los resultados, se encuentra en el artículo "Altas frecuencias en un problema stiff" relativo a un modelo de vibraciones de una cuerda con dos tramos de muy distinta densidad. Se trata del problema:

$$\varepsilon y'' + \lambda y = 0, \quad x \in (-1, 0),$$

$$y'' + \lambda y = 0, \quad x \in (0, 1),$$

$$y(0^-) = y(0^+), \quad \varepsilon y'(0^-) = y'(0^+),$$

$$y(0) = 0, \; y(1) = 0.$$

% En relación al método en diferencias finitas (diferencias centradas), para resolver un problema de contorno en [a,b], se introduce una partición del intervalo [a,b]: los puntos x_i en los que se va a aproximar la solución de y''-c(x)y=d(x), y(a)=0, y(b)=0. Dado N, se toma x_0=a, x_{N+1}=b , y en cada punto interior x_i (i=1,2,…N) se aproxima y'' por (y(x_i+h)-2y(x_i)+y(x_i-h))/h^2. {x_i}_{i=0}^{N+1} es "la discretización" de [0,1]. Así, escribiendo en la ecuación diferencial x_i en vez de x, se busca la aproximación de y(x) en x=x_i de tal manera que sea solución de un sistema de ecuaciones algebraico para una matriz tridiagonal simétrica. Se trata de un método de orden 2 (c y d funciones continuas en [a,b]).

```
function [T,v] = ffinitge(FunF1, FunFcn, n)
h=1/(n+1);
x=0+h:h:1-h;
a=2*ones(n,1);
b=-ones((n-1),1);
Ann=diag(a)+diag(b,1)+diag(b,-1);
c=feval(FunF1,x);
An=Ann+h^2*diag(c);
g=feval(FunFcn, x);
f=-h^2*g(:);
u1=An\f;
t=[0,x,1];
T=t(:);
v=[0;u1;0];
plot(T,v)
```

% El programa para aproximar la solución del problema de contorno
y''-c(x)y=d(x), y(0)=y(1)=0
está en el fichero ffinitge.m que lee N (el número de puntos de la partición), y las funciones c(x) y d(x) están guardadas a su vez en los ficheros FunFcn.m y funF1.m. Por ejemplo, para c(x)=x^2, se introduce en FunF1.m

```
function [ y ] = fmic(x)
y=x.^2;
```

% ffinitge.m nos proporciona el valor aproximado de la solución en los nodos de la partición y la gráfica de la solución

*% **Ejercicio 1:** Aproximamos numéricamente la solución del problema de contorno*
y''-x^2y=e^x, y(0)=0, y(1)=0,
que es única, y no se encuentra explícitamente con dsolve.
Definimos las funciones c(x)=x^2 y d(x)=exp(x) en las funciones Matlab fmic.m y fmid.m, por ejemplo

>> type ffinitge

>> type fmid

>> type fmic

% una manera de evaluar estas funciones definidas a través de ficheros es con feval o directamente como una función interna:

```
>> feval('fmid',2)

ans =

   7.3891

>> fmid(2)

ans =

   7.3891
```

% para una separación entre nodos h=0.1, ejecutamos

```
>> [T,v] = ffinitge('fmic','fmid', 9);[T,v]

ans =
         0         0
    0.1000   -0.0652
    0.2000   -0.1194
    0.3000   -0.1614
    0.4000   -0.1900
    0.5000   -0.2040
    0.6000   -0.2021
    0.7000   -0.1826
    0.8000   -0.1440
    0.9000   -0.0839
    1.0000         0
```

% T almacena la discretización del intervalo; v contiene la aproximación de la solución, de manera que si dibujamos los pares de puntos (T,v) tenemos una gráfica aproximada de la solución, aproximación que debe mejorar al aumentar n. Aumentamos n y ejecutamos, haciendo la gráfica en distintos colores

>> hold on

>> [T,v]=ffinitge('fmic','fmid',19);[T,v]

ans =

0	0
0.0500	-0.0339
0.1000	-0.0652
0.1500	-0.0938
0.2000	-0.1194
0.2500	-0.1421
0.3000	-0.1615
0.3500	-0.1776
0.4000	-0.1901
0.4500	-0.1991
0.5000	-0.2042
0.5500	-0.2053
0.6000	-0.2023
0.6500	-0.1948
0.7000	-0.1828
0.7500	-0.1660
0.8000	-0.1441
0.8500	-0.1169
0.9000	-0.0840
0.9500	-0.0452
1.0000	0

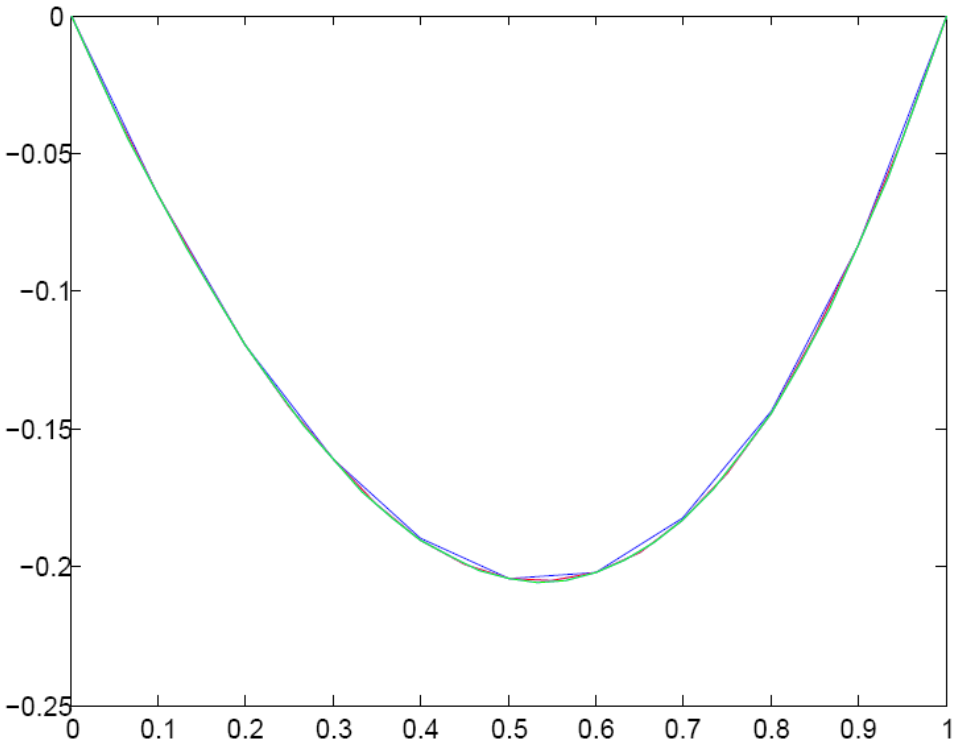
>> plot(T,v,'r')

>> [T,v] = ffinitge('fmic','fmid', 29);[T,v]

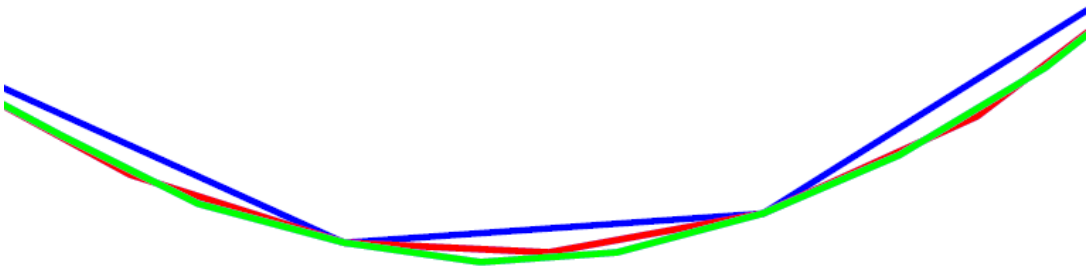
ans =

0	0
0.0333	-0.0229
0.0667	-0.0447
0.1000	-0.0653
0.1333	-0.0846
0.1667	-0.1027
0.2000	-0.1195
0.2333	-0.1349
0.2667	-0.1489
0.3000	-0.1615
0.3333	-0.1726
0.3667	-0.1822
0.4000	-0.1902
0.4333	-0.1965
0.4667	-0.2012
0.5000	-0.2042
0.5333	-0.2054
0.5667	-0.2048
0.6000	-0.2023
0.6333	-0.1978
0.6667	-0.1914
0.7000	-0.1828
0.7333	-0.1722
0.7667	-0.1593
0.8000	-0.1441
0.8333	-0.1266
0.8667	-0.1066
0.9000	-0.0840
0.9333	-0.0588
0.9667	-0.0309
1.0000	0

>> plot(T,v,'g')



% es evidente que la mejor aproximación debe ser la última, esto es, la gráfica en verde. Además, ampliando el dibujo cerca de las gráficas de las curvas, aparentemente superpuestas, vemos que hay una diferencia considerable entre los tres dibujadas.



% No obstante, para mejorar las aproximaciones ahorrando cálculos, al igual que pasa con las aproximaciones mediante Euler (eul.m) o mediante elementos finitos (elementosfinitos.m), y con otras, la función Mablab finitge.m debería contener tests de paradas que controlen, por ejemplo, el tamaño del paso para obtener una aproximación deseada, o para detectar variaciones rápidas de soluciones en determinados intervalos .

% La función Matlab bvp4c, para aproximar soluciones de problemas de contorno, nos da un error más pequeño que ffinitge.m y elementosfinitos.m, aunque la introducción de datos y la salida pueden resultar algo más complicados. La ventaja de esta función Matlab es que resuelve problemas para ecuaciones de orden superior, y para sistemas, con muy diversas condiciones de contorno.

% Resumimos las instrucciones que nos dan la gráfica aproximada de la solución del problema de contorno y''-x^2y=e^x, y(0)=0, y(1)=0

>> solinit=bvpinit(linspace(0,1,10),[1,0]);

>> sol=bvp4c(@ecuaorden2,@condcontorno,solinit);

>> plot(sol.x,sol.y(1,:),'*');

% analizamos abajo estas instrucciones

>> help bvp4c

```
BVP4C  Solve boundary value problems for ODEs by collocation.
   SOL = BVP4C(ODEFUN,BCFUN,SOLINIT) integrates a system of ordinary
   differential equations of the form y' = f(x,y) on the interval [a,b],
   subject to general two-point boundary conditions of the form
   bc(y(a),y(b)) = 0. ODEFUN and BCFUN are function handles. For a scalar X
   and a column vector Y, ODEFUN(X,Y) must return a column vector representing
   f(x,y). For column vectors YA and YB, BCFUN(YA,YB) must return a column
   vector representing bc(y(a),y(b)).  SOLINIT is a structure with fields
      x -- ordered nodes of the initial mesh with
           SOLINIT.x(1) = a, SOLINIT.x(end) = b
      y -- initial guess for the solution with SOLINIT.y(:,i)
           a guess for y(x(i)), the solution at the node SOLINIT.x(i)

   BVP4C produces a solution that is continuous on [a,b] and has a
   continuous first derivative there. The solution is evaluated at points
   XINT using the output SOL of BVP4C and the function DEVAL:
   YINT = DEVAL(SOL,XINT). The output SOL is a structure with
      SOL.x  -- mesh selected by BVP4C
      SOL.y  -- approximation to y(x) at the mesh points of SOL.x
      SOL.yp -- approximation to y'(x) at the mesh points of SOL.x
      SOL.solver -- 'bvp4c'

   SOL = BVP4C(ODEFUN,BCFUN,SOLINIT,OPTIONS) solves as above with default
   parameters replaced by values in OPTIONS, a structure created with the
   BVPSET function. To reduce the run time greatly, use OPTIONS to supply
   a function for evaluating the Jacobian and/or vectorize ODEFUN.
   See BVPSET for details and SHOCKBVP for an example that does both.

   Some boundary value problems involve a vector of unknown parameters p
   that must be computed along with y(x):
      y' = f(x,y,p)
      0  = bc(y(a),y(b),p)
   For such problems the field SOLINIT.parameters is used to provide a guess
   for the unknown parameters. On output the parameters found are returned
   in the field SOL.parameters. The solution SOL of a problem with one set
   of parameter values can be used as SOLINIT for another set. Difficult BVPs
   may be solved by continuation: start with parameter values for which you can
   get a solution, and use it as a guess for the solution of a problem with
```

```
parameters closer to the ones you want. Repeat until you solve the BVP
for the parameters you want.

The function BVPINIT forms the guess structure in the most common
situations:  SOLINIT = BVPINIT(X,YINIT) forms the guess for an initial
mesh X as described for SOLINIT.x, and YINIT either a constant vector
guess for the solution or a function handle. If YINIT is a function handle
then for a scalar X, YINIT(X) must return a column vector, a guess for
the solution at point x in [a,b]. If the problem involves unknown parameters
SOLINIT = BVPINIT(X,YINIT,PARAMS) forms the guess with the vector PARAMS of
guesses for the unknown parameters.

BVP4C solves a class of singular BVPs, including problems with
unknown parameters p, of the form
   y' = S*y/x + f(x,y,p)
   0  = bc(y(0),y(b),p)
The interval is required to be [0, b] with b > 0.
Often such problems arise when computing a smooth solution of
ODEs that result from PDEs because of cylindrical or spherical
symmetry. For singular problems the (constant) matrix S is
specified as the value of the 'SingularTerm' option of BVPSET,
and ODEFUN evaluates only f(x,y,p). The boundary conditions
must be consistent with the necessary condition S*y(0) = 0 and
the initial guess should satisfy this condition.

BVP4C can solve multipoint boundary value problems.  For such problems
there are boundary conditions at points in [a,b]. Generally these points
represent interfaces and provide a natural division of [a,b] into regions.
BVP4C enumerates the regions from left to right (from a to b), with indices
starting from 1.  In region k, BVP4C evaluates the derivative as
YP = ODEFUN(X,Y,K).  In the boundary conditions function,
BCFUN(YLEFT,YRIGHT), YLEFT(:,K) is the solution at the 'left' boundary
of region k and similarly for YRIGHT(:,K).  When an initial guess is
created with BVPINIT(XINIT,YINIT), XINIT must have double entries for
each interface point. If YINIT is a function handle, BVPINIT calls
Y = YINIT(X,K) to get an initial guess for the solution at X in region k.
In the solution structure SOL returned by BVP4C, SOL.x has double entries
for each interface point. The corresponding columns of SOL.y contain
the 'left' and 'right' solution at the interface, respectively.
See THREEBVP for an example of solving a three-point BVP.

Example
     solinit = bvpinit([0 1 2 3 4],[1 0]);
     sol = bvp4c(@twoode,@twobc,solinit);
  solve a BVP on the interval [0,4] with differential equations and
  boundary conditions computed by functions twoode and twobc, respectively.
  This example uses [0 1 2 3 4] as an initial mesh, and [1 0] as an initial
  approximation of the solution components at the mesh points.
     xint = linspace(0,4);
     yint = deval(sol,xint);
  evaluate the solution at 100 equally spaced points in [0 4]. The first
  component of the solution is then plotted with
     plot(xint,yint(1,:));
For more examples see TWOBVP, FSBVP, SHOCKBVP, MAT4BVP, EMDENBVP, THREEBVP.

See also bvp5c, bvpset, bvpget, bvpinit, bvpxtend, deval, function_handle.
```

% pinchando en los enlaces se puede tener información sobre bvpinit, por ejemplo, pero también con help:

>> help bvpinit

```
BVPINIT  Form the initial guess for BVP solvers.
   SOLINIT = BVPINIT(X,YINIT) forms the initial guess for BVP4C in common
   circumstances. The boundary value problem (BVP) is to be solved on [a,b].
   The vector X specifies a and b as X(1) = a and X(end) = b. It is also
   a guess for an appropriate mesh. BVP4C will adapt this mesh to the solution,
   so often a guess like X = linspace(a,b,10) will suffice, but in difficult
   cases, mesh points should be placed where the solution changes rapidly.

   The entries of X must be ordered. For two-point BVPs, the entries of X
   must be distinct, so if a < b, then X(1) < X(2) < ... < X(end), and
   similarly for a > b. For multipoint BVPs there are boundary conditions
   at points in [a,b]. Generally, these points represent interfaces and
   provide a natural division of [a,b] into regions. BVPINIT enumerates
   the regions from left to right (from a to b), with indices starting
   from 1. You can specify interfaces by double entries in the initial
   mesh X. BVPINIT interprets one entry as the right end point of region k
   and the other as the left end point of region k+1. THREEBVP exemplifies
   this for a three-point BVP.

   YINIT provides a guess for the solution. It must be possible to evaluate
   the differential equations and boundary conditions for this guess.
   YINIT can be either a vector or a function handle:

   vector:  YINIT(i) is a constant guess for the i-th component Y(i,:) of
            the solution at all the mesh points in X.

   function:  YINIT is a function of a scalar x. For example, use
              solinit = bvpinit(x,@yfun) if for any x in [a,b], yfun(x)
              returns a guess for the solution y(x). For multipoint BVPs,
              BVPINIT calls Y = YINIT(X,K) to get an initial guess for the
              solution at x in region k.

   SOLINIT = BVPINIT(X,YINIT,PARAMETERS) indicates that the BVP involves
   unknown parameters. A guess must be provided for all parameters in the
   vector PARAMETERS.

   See also bvpget, bvpset, bvp4c, bvp5c, bvpxtend, deval, function_handle.

   Reference page in Help browser
      doc bvpinit
```

% bvp4c usa una discretización del intervalo en que se resuelve el problema de contorno (intervalo y discretización que hay que indicar con la función bvpinit), y un método de colocación que implica la resolución de un sistema. Dicho sistema es en general no lineal, se resuelve iterativamente; y para esto también hay que introducir una pista inicial en bvpinit (valores iniciales). Para más detalles, ver las direcciones: http://www.mathworks.es/es/help/matlab/ref/bvp4c.html y http://www.mathworks.com/matlabcentral/answers/uploaded_files/1113/bvp_paper.pdf

*% La función lineal (asociada a y''= exp(x) +x^2y) a introducir en una función Matlab, para utilizar con bvp4c, está dada en forma vectorial y1=y, y2=y' (esto es, se introduce la ED como el sistema y1'=y2, y2'=exp(x)+x^2*y1 en ecuaorden2.m)*

```
function dydx=ecuaorden2(x,y)
dydx=[y(2)
      +x^2*y(1)+exp(x)];
```

% Las condiciones de contorno son entonces y1(0)=0, y1(1)=0, y esto se introduce en la función Matlab condcontorno.m

```
function res=condcontorno(ya,yb)
res=[ya(1)
     yb(1)];
```

% De manera general, se introduce en el primer vector y(a) (o y'(a) o una relación entre ambas) menos la condición impuesta en el punto x=a; en el segundo vector y(b) (o y'(b) o una relación entre ambas) menos la condición impuesta en x=b.

>> type ecuaorden2

>> type condcontorno

% una vez creados los ficheros con la ecuación y las condiciones de contorno, se usa solinit=bvpinit(vector discretizado de (a,b), condiciones iniciales tests).

```
>> solinit=bvpinit(linspace(0,1,10),[1,0])

solinit =

   solver: 'bvpinit'
        x: [0 0.1111 0.2222 0.3333 0.4444 0.5556 0.6667 0.7778 0.8889 1]
        y: [2x10 double]
    yinit: [1 0]

>> sol=bvp4c(@ecuaorden2,@condcontorno,solinit)

sol =

   solver: 'bvp4c'
        x: [0 0.1111 0.2222 0.3333 0.4444 0.5556 0.6667 0.7778 0.8889 1]
        y: [2x10 double]
       yp: [2x10 double]
    stats: [1x1 struct]
```

% sol.x contiene la partición del intervalo [0,1], mientras que sol.y contiene la solución aproximada en los puntos sol.x (sol.y(1,:) contiene la aproximación de la solución, y sol.y(2,:) la de la derivada)

>> [sol.x;sol.y]

ans =

Columns 1 through 8

0	0.1111	0.2222	0.3333	0.4444	0.5556	0.6667	0.7778
0	-0.0718	-0.1299	-0.1726	-0.1983	-0.2052	-0.1914	-0.1545
-0.7043	-0.5868	-0.4558	-0.3104	-0.1496	0.0280	0.2245	0.4433

Columns 9 through 10

0.8889	1.0000
-0.0919	0
0.6895	0.9707

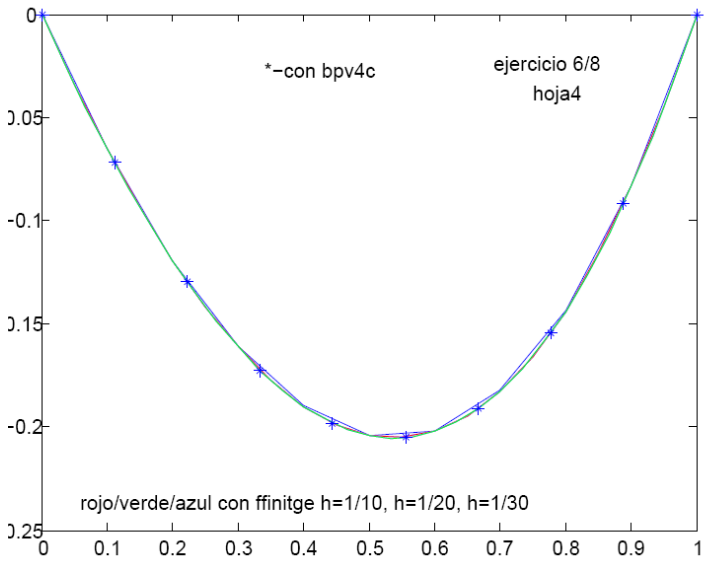
>> plot(sol.x,sol.y(1,:),'*')

>> gtext('*-con bpv4c')

>> gtext('rojo/verde/azul con ffinitge h=1/10, h=1/20, h=1/30')

>> gtext('ejercicio 6/8') *% comentario orientativo que se puede cambiar*

>> gtext('hoja4') *% comentario orientativo que se puede cambiar*



% A continuación comparamos la solución obtenida con diferencias finitas y elementos finitos (fichero elementosfinitosnoresol.m) para el tamaño del paso h=0.1

>> [t,y]= ffinitge('fmic','fmid',9);[t,y]

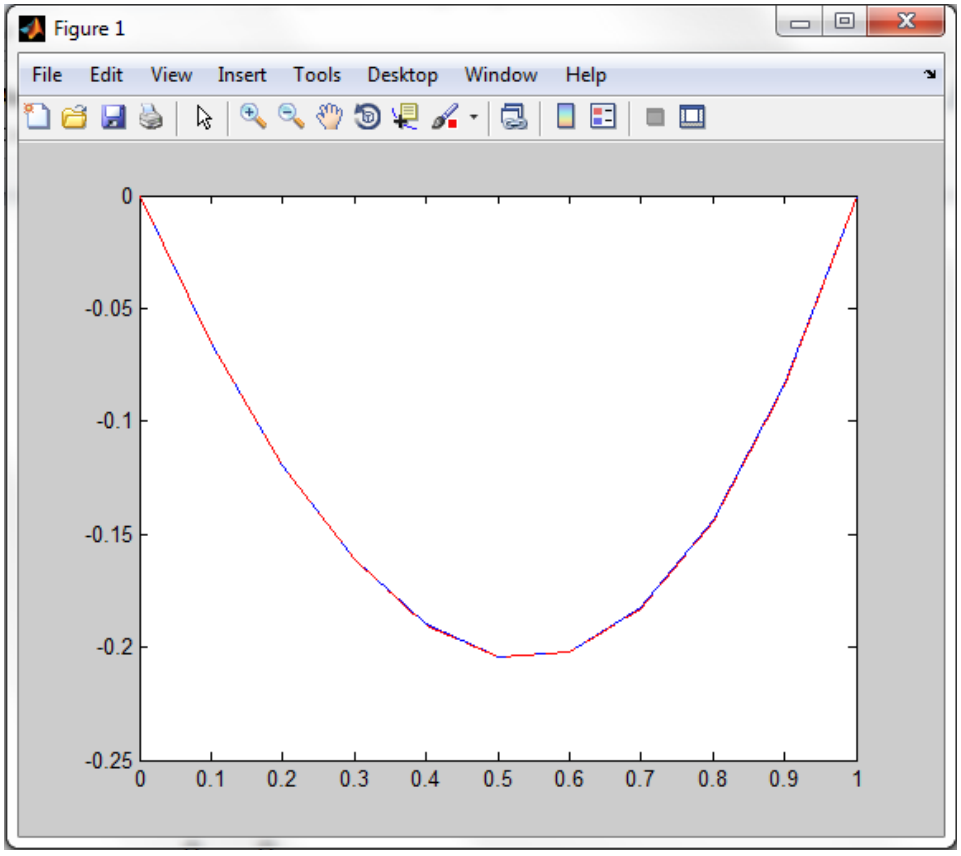
```
ans =

      0       0
 0.1000  -0.0652
 0.2000  -0.1194
 0.3000  -0.1614
 0.4000  -0.1900
 0.5000  -0.2040
 0.6000  -0.2021
 0.7000  -0.1826
 0.8000  -0.1440
 0.9000  -0.0839
 1.0000        0
```

>> hold on

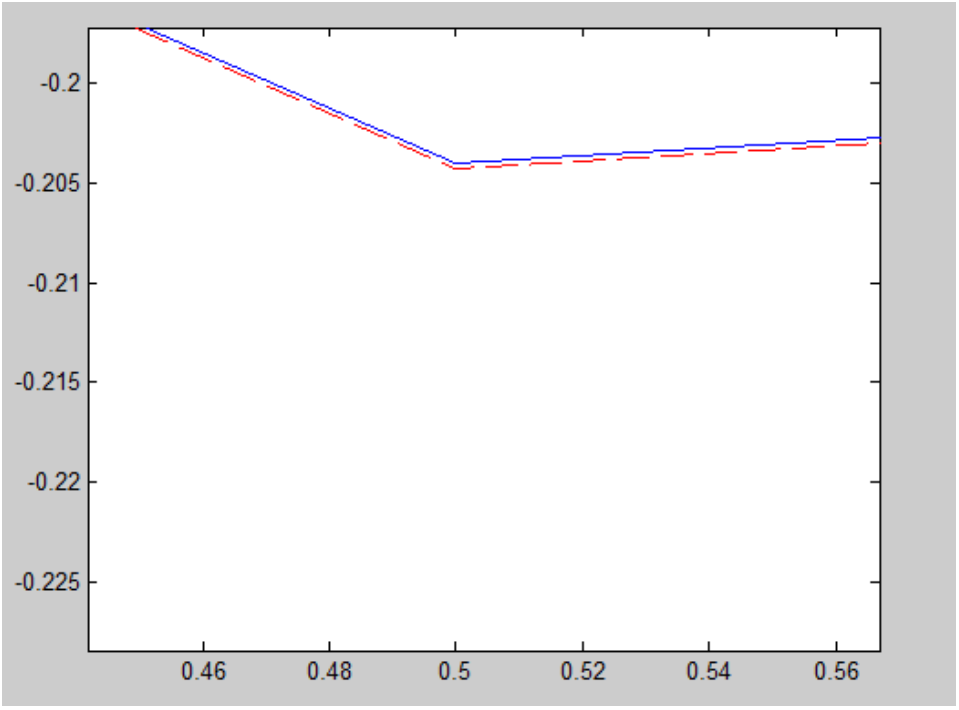
>> elementosfinitosnoresol(9)

```
nodos_y_aproximacion =

      0       0
 0.1000  -0.0653
 0.2000  -0.1195
 0.3000  -0.1615
 0.4000  -0.1902
 0.5000  -0.2043
 0.6000  -0.2024
 0.7000  -0.1829
 0.8000  -0.1442
 0.9000  -0.0841
 1.0000        0
```

% vemos que hay diferencia de aproximación en cada nodo en la tercera o cuarta cifra decimal, aunque las gráficas de las aproximaciones prácticamente coinciden (en rojo la obtenida con el método de los elementos finitos)



% una ampliación (con el editor gráfico) de parte de la gráfica nos muestra que ambas gráficas no coinciden



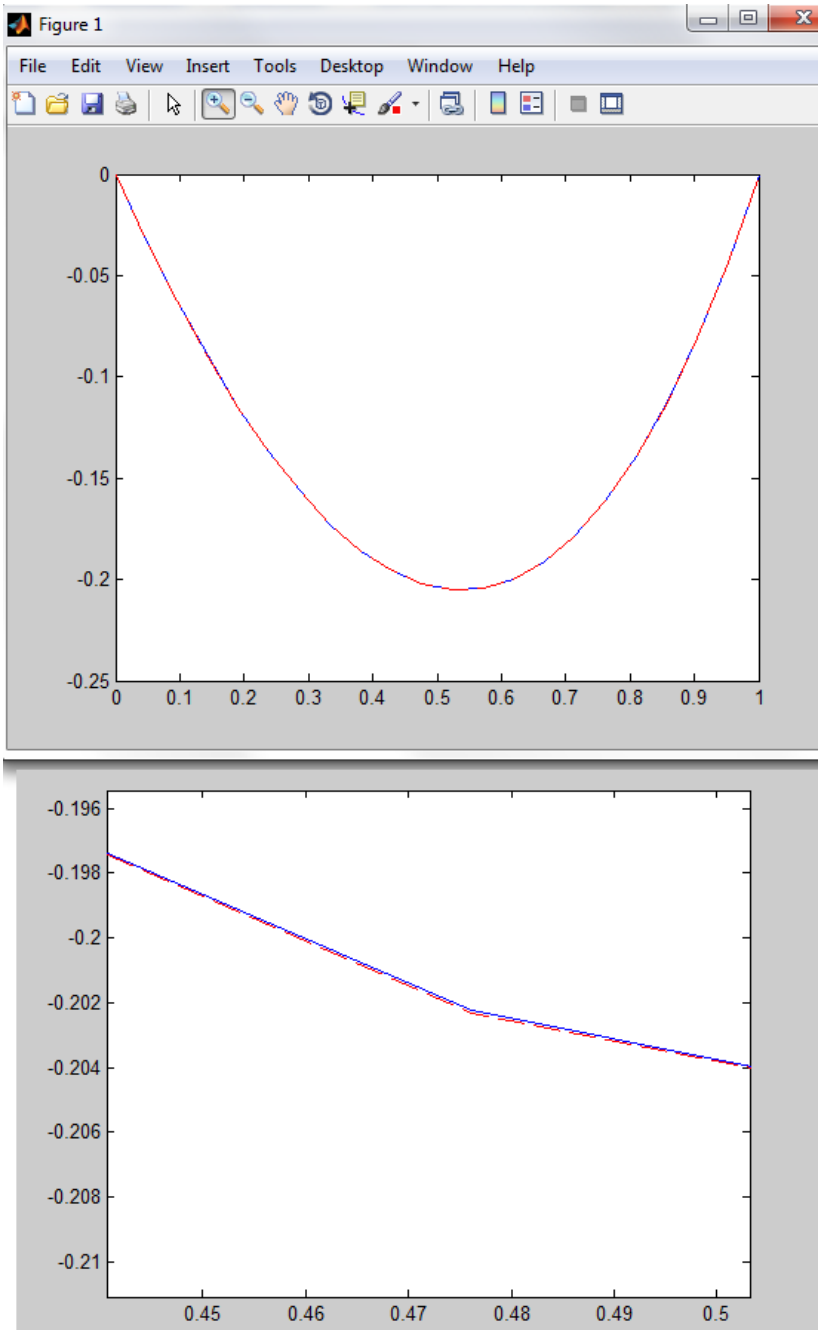
% Aumentando n en ambas funciones Matlab se tiene mejor aproximación y una gráfica menos angulosa

>> hold off

>> [t,y]= ffinitge('fmic','fmid',20);

>> hold on

>> elementosfinitosnoresol(20);

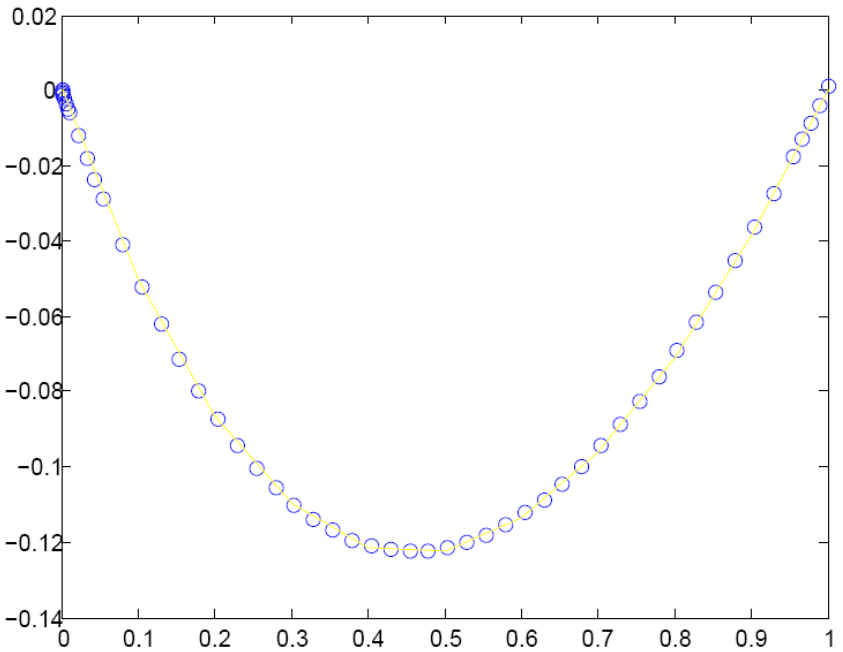


*% **Ejercicio 3**, **problema no lineal**: y''=x^3y^2+y'-1, y(0)=0, y(1)=0*
Aproximamos la solución con bvpc4 y comparamos con el método de tiro.

% Hacemos primero la gráfica de la solución con el método de tiro sólo (una vez que se ha ajustado la derivada a introducir para que y(1)=0). Utilizamos ode45 y la función ftirocomp.m que se ha introducido para resolver un problema de Cauchy para sistemas, y ajustamos la condición inicial y'(0)=beta para que y(1)=0. Se comprueba, testeando varios valores de beta, que beta está cerca de -0.58 (esto es, y(0)=0, y'(0)=-0.58 implica y(1) aproximado a 0 con un error)

```
function dydx=ftirocomp(x,y)
dydx = [y(2); -(y(1))^2*x^3+1-y(2)];
```

```
>> [t,y]=ode45('ftirocomp',[0,1],[0,-0.58]);

>> plot(t,y(:,1),'o')

>> hold on

>> plot(t,y(:,1),'y')

>> hold off
```



% Para resolver el problema utilizando bvp4c, comenzamos definiendo solinit que proporciona valores iniciales para los dos vectores y(1,:) e y(2,:), necesarios para que se empiece el proceso iterativo: en este caso, son valores constantes 1 y 0 respectivamente

>> solinit=bvpinit(linspace(0,1,10),[1,0])

```
solinit =

   solver: 'bvpinit'
        x: [0 0.1111 0.2222 0.3333 0.4444 0.5556 0.6667 0.7778 0.8889 1]
        y: [2x10 double]
    yinit: [1 0]
```

% arriba se han considerado 10 puntos entre [0,1]; para ajustar el tamaño del paso a h=0.1 hay que tomar 11 puntos entre [0,1]:

>> solinit=bvpinit(linspace(0,1,11),[1,0])

```
solinit =

   solver: 'bvpinit'
        x: [0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1]
        y: [2x11 double]
    yinit: [1 0]
```

% para ver la condición inicial proporcionada:

>> solinit.y

```
ans =

    1    1    1    1    1    1    1    1    1    1    1
    0    0    0    0    0    0    0    0    0    0    0
```

% las condiciones de contorno ese introducen en el fichero condcontorno.m y la ecuación diferencial en forma vectorial (el sistema y'_1=y_2, y'_2= -(x^3y_1^2+y_2-1)) en el fichero ecuaorden2nolineal.m

>> type ecuaorden2nolineal

% se aproxima la solución del problema de contorno

>> sol=bvp4c(@ecuaorden2nolineal,@condcontorno,solinit)

```
sol =

   solver: 'bvp4c'
        x: [0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1]
        y: [2x11 double]
       yp: [2x11 double]
    stats: [1x1 struct]
```

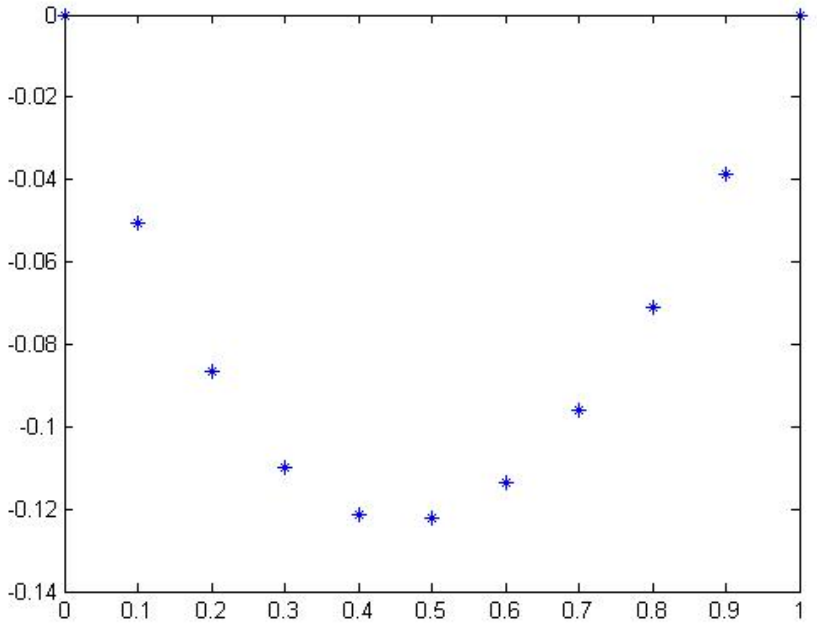
% De los datos que proporciona bvp4c, x almacena la partición tomada, y almacena la aproximación de la solución y de la derivada, esto es las componentes del vector solución . Para visualizar estos datos lo hacemos con sol.x, sol.y. Si tenemos una ecuación diferencial de segundo orden, con sol.y(1,:) sol.y(2,:), obtenemos la aproximación en los puntos x de la solución y derivada respectivamente

>> [sol.x;sol.y]

```
ans =

  Columns 1 through 8

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000
         0   -0.0505   -0.0867   -0.1099   -0.1214   -0.1222   -0.1136   -0.0962
   -0.5814   -0.4309   -0.2947   -0.1715   -0.0601    0.0406    0.1317    0.2140

  Columns 9 through 11

    0.8000    0.9000    1.0000
   -0.0710   -0.0387         0
    0.2885    0.3561    0.4173
```

% dibujando los pares de puntos se tiene la gráfica de la solución

>> plot(sol.x,sol.y(1,:),'*')



% A continuación, comparamos la aproximación por bvp4c con la obtenida por el método de tiro:

>> type ftirocomp

>> [t,y]=ode45('ftirocomp',[0,1],[0,-0.58]);[t,y]

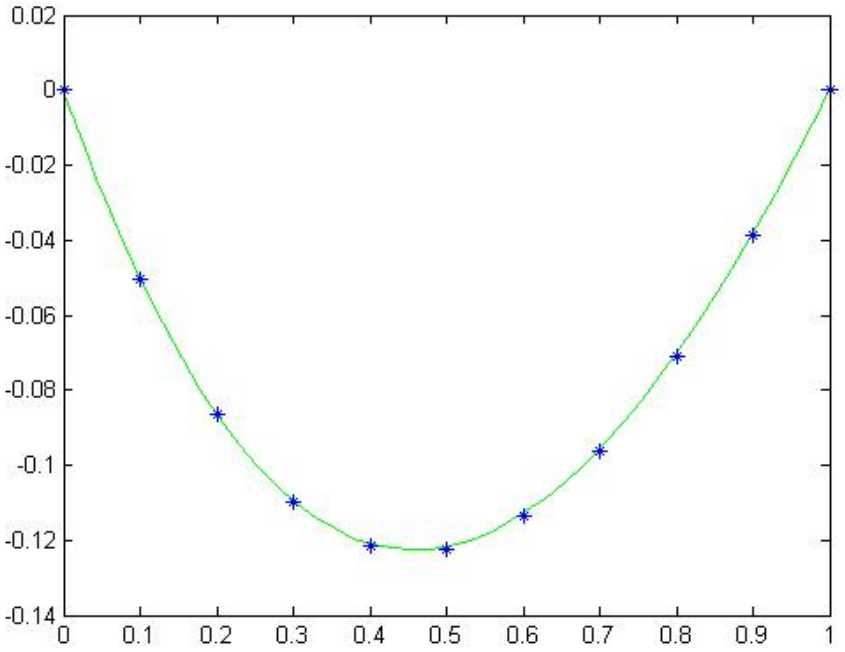
ans =

0	0	-0.5800
0.0001	-0.0001	-0.5799
0.0002	-0.0001	-0.5797
0.0003	-0.0002	-0.5796
0.0003	-0.0002	-0.5795
0.0008	-0.0005	-0.5788
0.0012	-0.0007	-0.5781
0.0016	-0.0010	-0.5774
0.0021	-0.0012	-0.5767
0.0042	-0.0024	-0.5733
0.0064	-0.0037	-0.5699
0.0086	-0.0049	-0.5665
0.0107	-0.0061	-0.5631
0.0216	-0.0121	-0.5463
0.0324	-0.0180	-0.5296

0.0432	-0.0236	-0.5132
0.0540	-0.0291	-0.4969
0.0790	-0.0410	-0.4599
0.1040	-0.0521	-0.4239
0.1290	-0.0622	-0.3887
0.1540	-0.0715	-0.3544
0.1790	-0.0800	-0.3210
0.2040	-0.0876	-0.2884
0.2290	-0.0944	-0.2566
0.2540	-0.1004	-0.2255
0.2790	-0.1057	-0.1953
0.3040	-0.1102	-0.1658
0.3290	-0.1140	-0.1370
0.3540	-0.1170	-0.1089
0.3790	-0.1194	-0.0816
0.4040	-0.1211	-0.0549
0.4290	-0.1222	-0.0289
0.4540	-0.1226	-0.0035
0.4790	-0.1224	0.0212
0.5040	-0.1215	0.0453
0.5290	-0.1201	0.0689
0.5540	-0.1181	0.0918
0.5790	-0.1155	0.1142
0.6040	-0.1124	0.1360
0.6290	-0.1087	0.1572
0.6540	-0.1045	0.1780
0.6790	-0.0998	0.1982
0.7040	-0.0946	0.2179
0.7290	-0.0889	0.2371
0.7540	-0.0828	0.2559
0.7790	-0.0761	0.2742
0.8040	-0.0691	0.2921
0.8290	-0.0615	0.3095
0.8540	-0.0536	0.3265
0.8790	-0.0452	0.3431
0.9040	-0.0364	0.3593
0.9290	-0.0273	0.3751
0.9540	-0.0177	0.3905
0.9655	-0.0132	0.3974
0.9770	-0.0086	0.4043
0.9885	-0.0039	0.4111
1.0000	0.0009	0.4178

>> hold on *% comparando gráficas*

>> plot(t,y(:,1),'g')



% la misma gráfica en distintos colores y formato: en verde con ode45, y la otra con bvp4c

*% **Ejercicio 6:** un método en diferencias finitas para el cálculo numérico de los valores propios y funciones propias en un problema de contorno en (-1,1) que se obtiene por separación de variables en el modelo de vibraciones de una cuerda que tiene una parte muy rígida comparada con la otra. Los extremos de la cuerda x=-1 y x=1 se suponen fijos. En la función Matlab fvpstiff.m, el parámetro **e** mide la rigidez (de la parte menos rígida) mientras que **n** está relacionado con la discretización del intervalo (1,1). A medida que **e** se hace más pequeño, se observa que las vibraciones de bajas frecuencias afectan fundamentalmente a la parte menos rígida, y es necesario considerar las altas frecuencias para la descripción completa de las vibraciones de la cuerda.*

```
% 2n+1 es el tamaño de la partición en [-1,1]

% e es la rigidez, que vamos a hacer tender a cero

% ey''=- \lambda y, en (-1,0)

% y''=-lambday en (0,1); y(-1)=0,y(1)=0.
```

```
>> type fvpstiff
```

```
>> fvpstiff(51,0.01) % h=1/52, e=0.01

valorpropio =

    0.0932

valorpropio =

    0.3723


valorpropio =

    0.8348


valorpropio =

    1.4684

valorpropio =

    2.1863
```

valorpropio =

2.7384

valorpropio =

3.5790

valorpropio =

4.7889

valorpropio =

6.2365

valorpropio =

7.9026

valorpropio =

9.7867

valorpropio =

11.8915

valorpropio =

14.2192

valorpropio =

16.7653

valorpropio =

19.4901

valorpropio =

22.0687

valorpropio =

23.7983

valorpropio =

26.4960

valorpropio =

29.9981

valorpropio =

33.8778

valorpropio =

38.0824

valorpropio =

42.6072

valorpropio =

47.4533

valorpropio =

52.6109

valorpropio =

57.9982

valorpropio =

62.8301

valorpropio =

65.8601

valorpropio =

71.1937

valorpropio =

77.7615

valorpropio =

84.8664

valorpropio =

92.4286

valorpropio =

100.4275

valorpropio =

108.8318

valorpropio =

117.4884

valorpropio =

124.7555

valorpropio =

129.1845

valorpropio =

137.8825

valorpropio =

147.9795

valorpropio =

158.6222

valorpropio =

169.6809

valorpropio =

181.0739

valorpropio =

192.6699

valorpropio =

203.8299

valorpropio =

209.7757

valorpropio =

217.9970

valorpropio =

229.7516

valorpropio =

241.7383

valorpropio =

253.5754

valorpropio =

265.0680

valorpropio =

276.0358

valorpropio =

286.2952

valorpropio =

295.6561

valorpropio =

303.9120

valorpropio =

310.6689

valorpropio =

313.6085

valorpropio =

317.1502

valorpropio =

321.1383

valorpropio =

323.6364

valorpropio =

436.7505

valorpropio =

582.9379

valorpropio =

751.7050

valorpropio =

943.6015

valorpropio =

1.1593e+003

valorpropio =

1.3996e+003

valorpropio =

1.6654e+003

valorpropio =

1.9577e+003

valorpropio =

2.2774e+003

valorpropio =

2.6258e+003

valorpropio =

3.0040e+003

valorpropio =

3.4134e+003

valorpropio =

3.8554e+003

valorpropio =

4.3315e+003

valorpropio =

4.8431e+003

valorpropio =

5.3919e+003

valorpropio =

5.9795e+003

valorpropio =

6.6074e+003

valorpropio =

7.2774e+003

```
valorpropio =

  7.9908e+003


valorpropio =

  8.7493e+003


valorpropio =

  9.5539e+003


valorpropio =

  1.0406e+004


valorpropio =

  1.1306e+004


valorpropio =

  1.2254e+004


valorpropio =

  1.3250e+004


valorpropio =

  1.4293e+004


valorpropio =

  1.5382e+004
```

valorpropio =

1.6512e+004

valorpropio =

1.7682e+004

valorpropio =

1.8884e+004

valorpropio =

2.0112e+004

valorpropio =

2.1358e+004

valorpropio =

2.2611e+004

valorpropio =

2.3859e+004

valorpropio =

2.5087e+004

valorpropio =

2.6279e+004

valorpropio =

2.7419e+004

valorpropio =

2.8486e+004

valorpropio =

2.9462e+004

valorpropio =

3.0329e+004

valorpropio =

3.1067e+004

valorpropio =

3.1660e+004

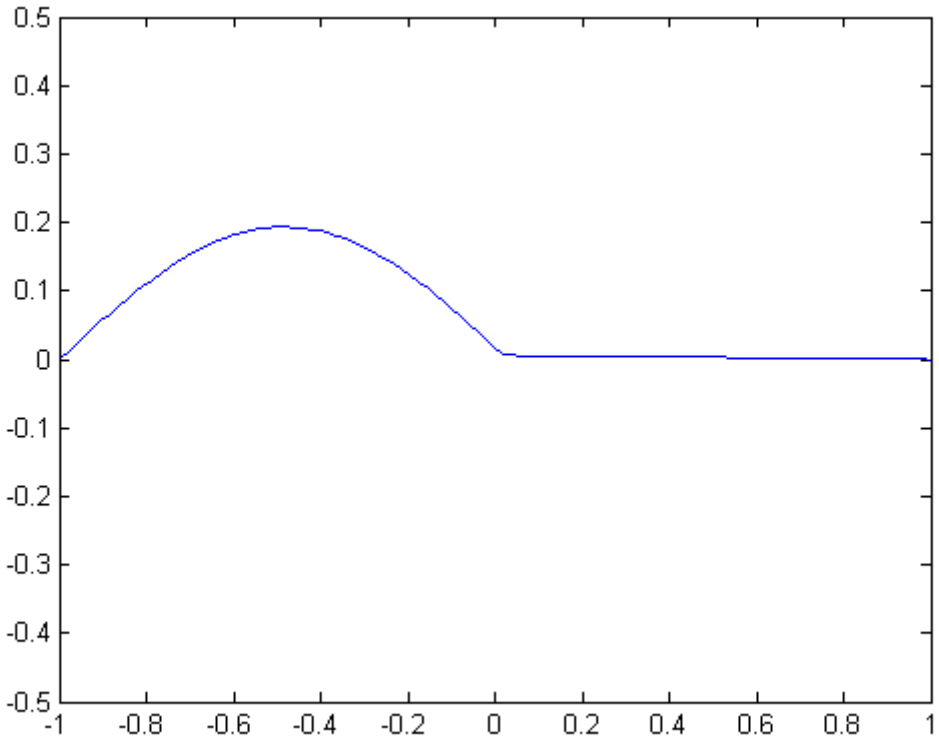
valorpropio =

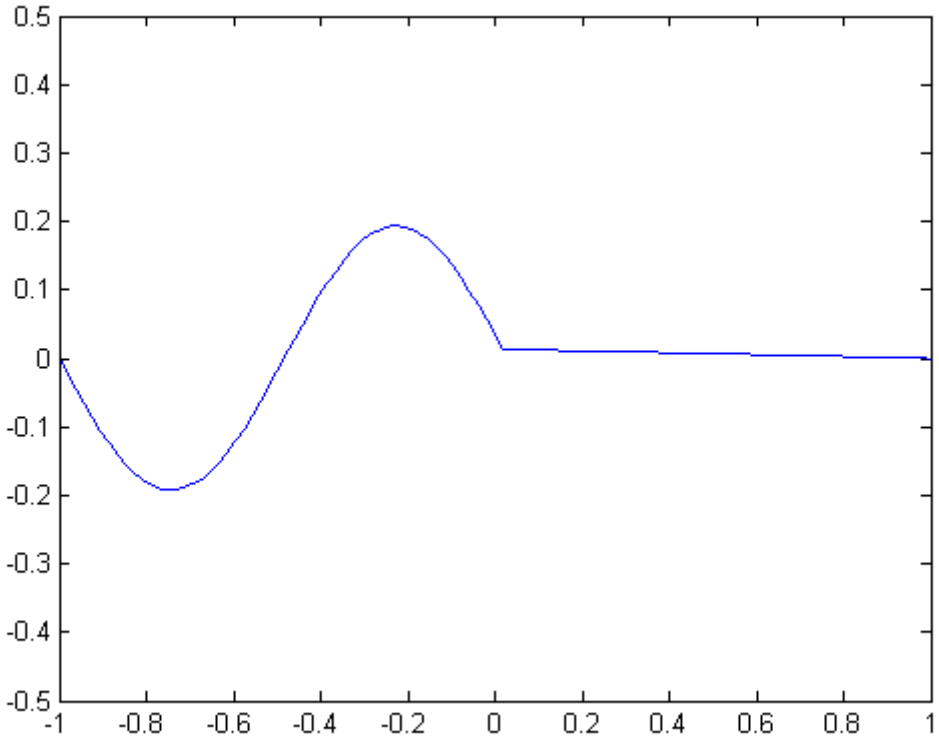
3.2094e+004

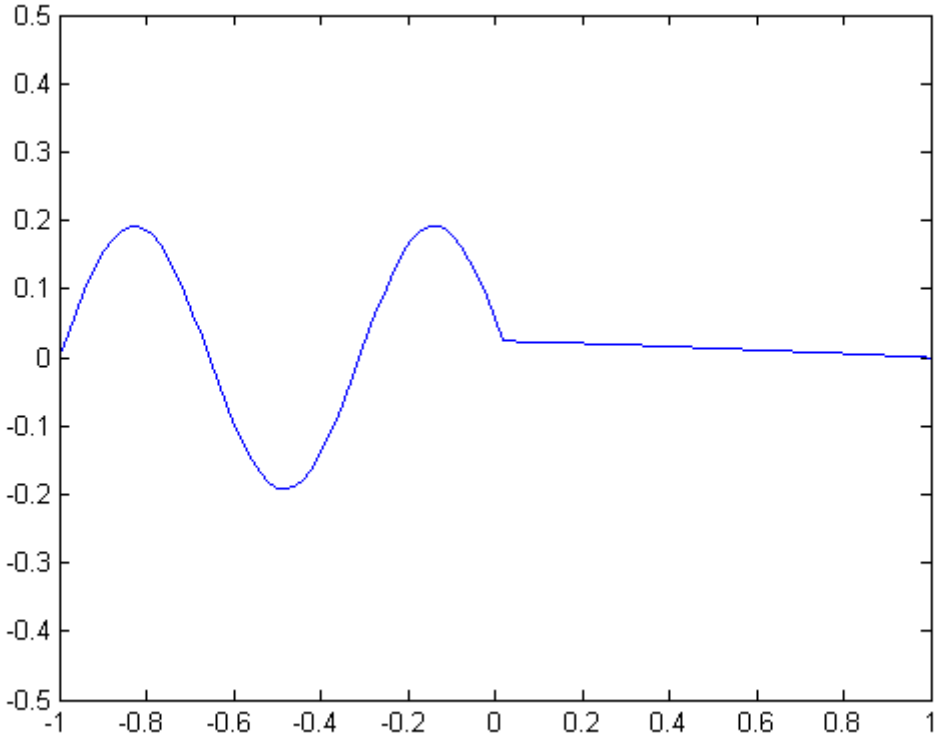
valorpropio =

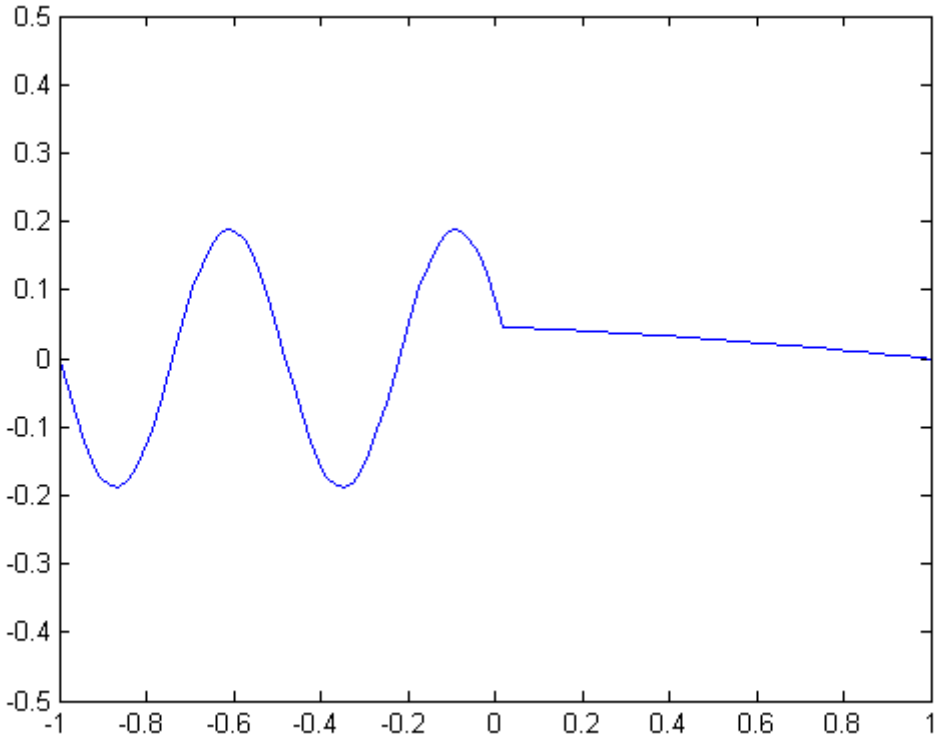
3.2359e+004

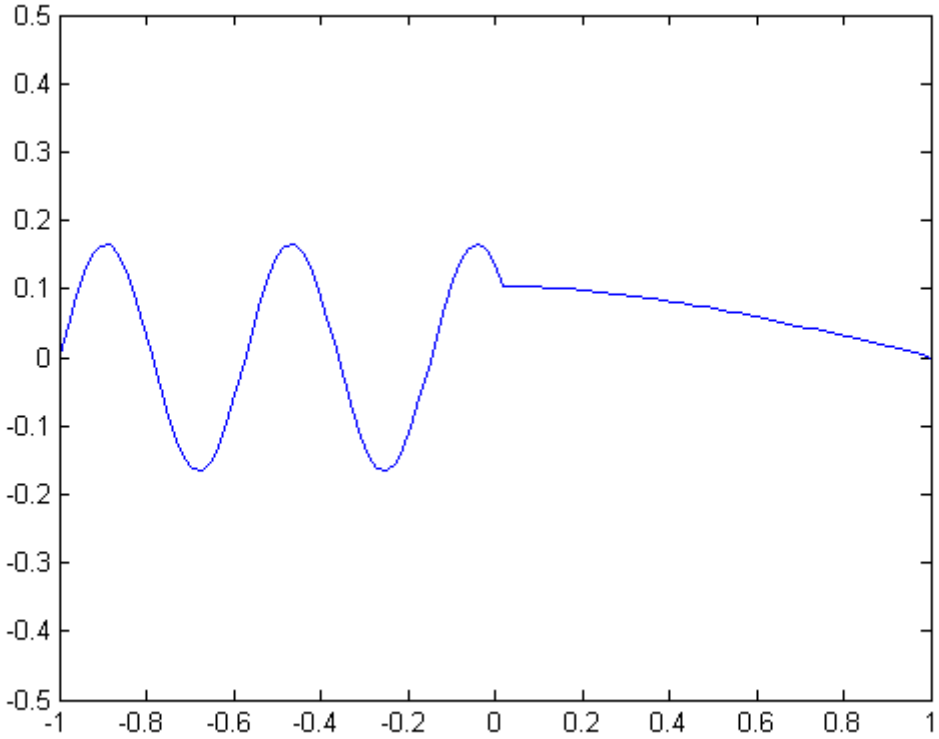
% A continuación se tienen las gráficas de las funciones propias asociadas a estos valores propios.

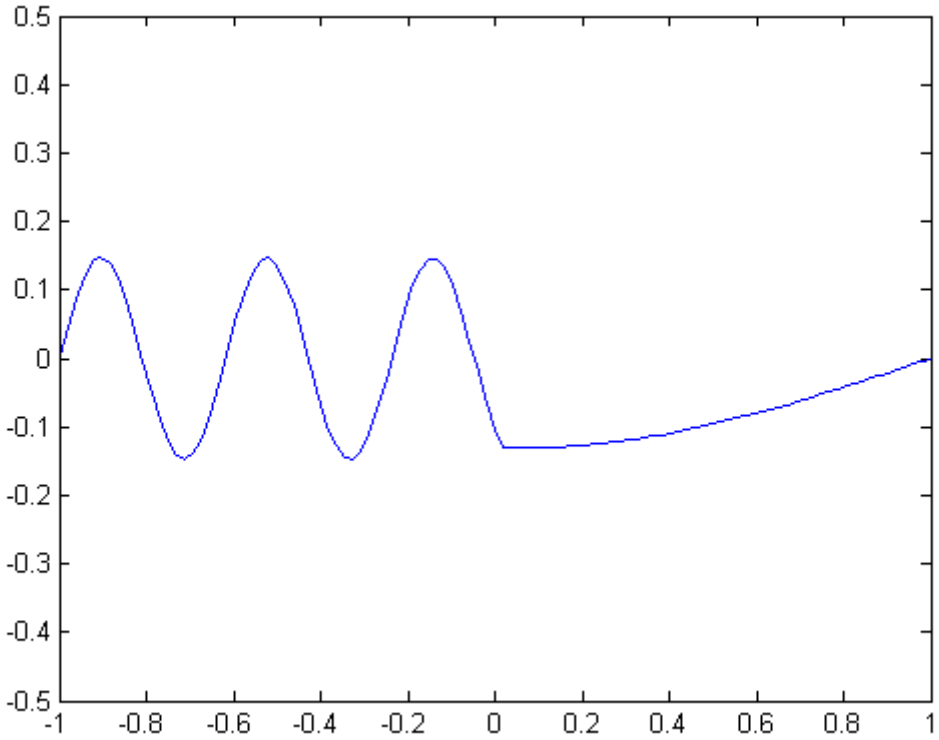

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

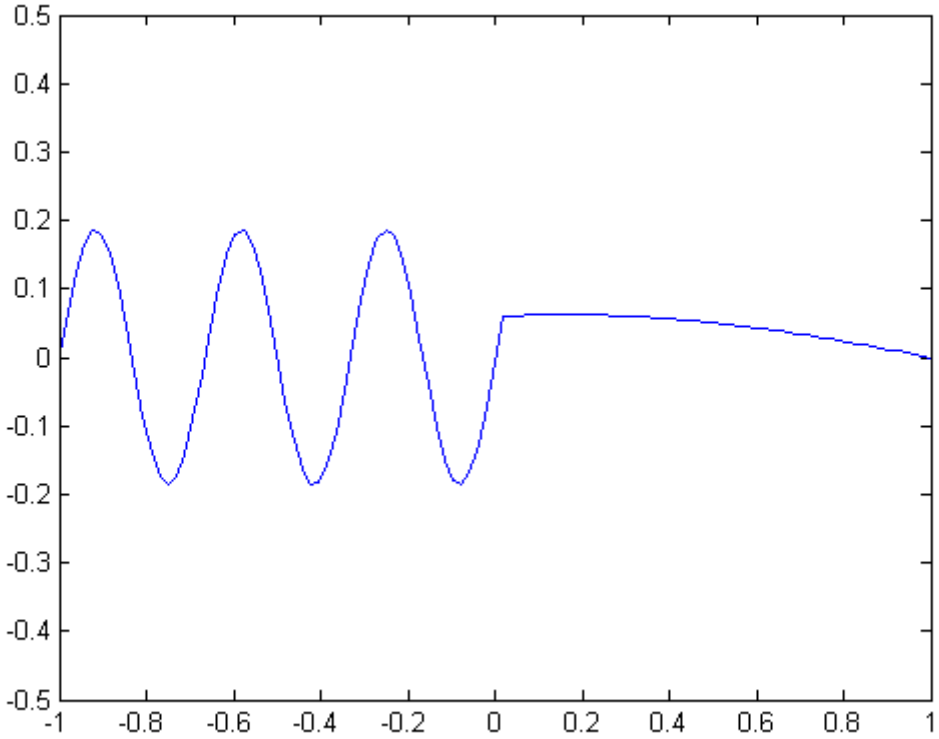

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

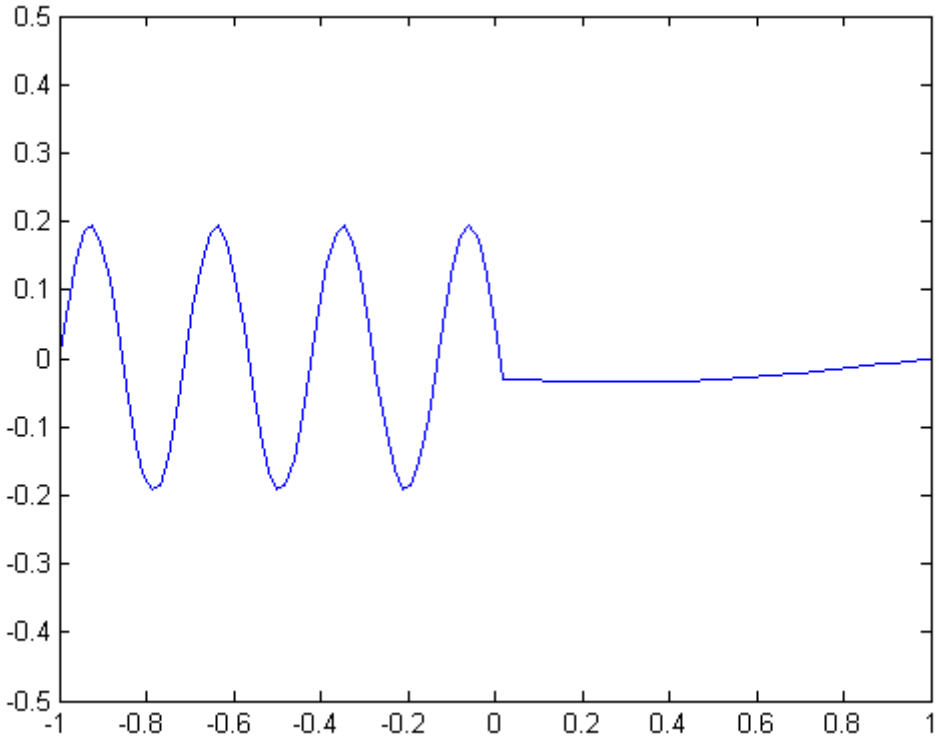

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

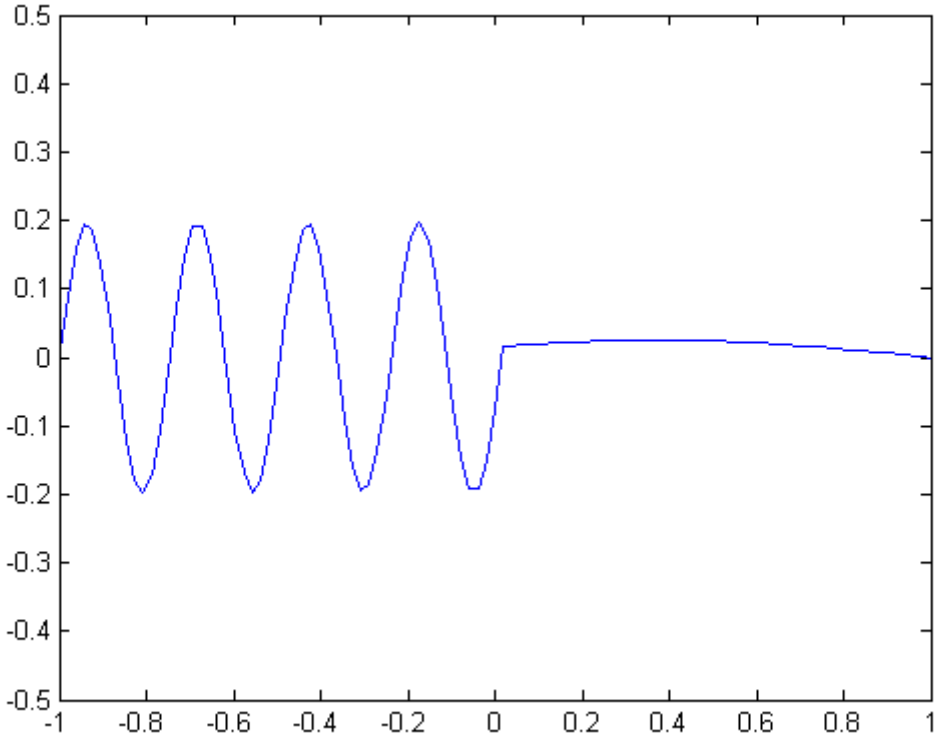

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

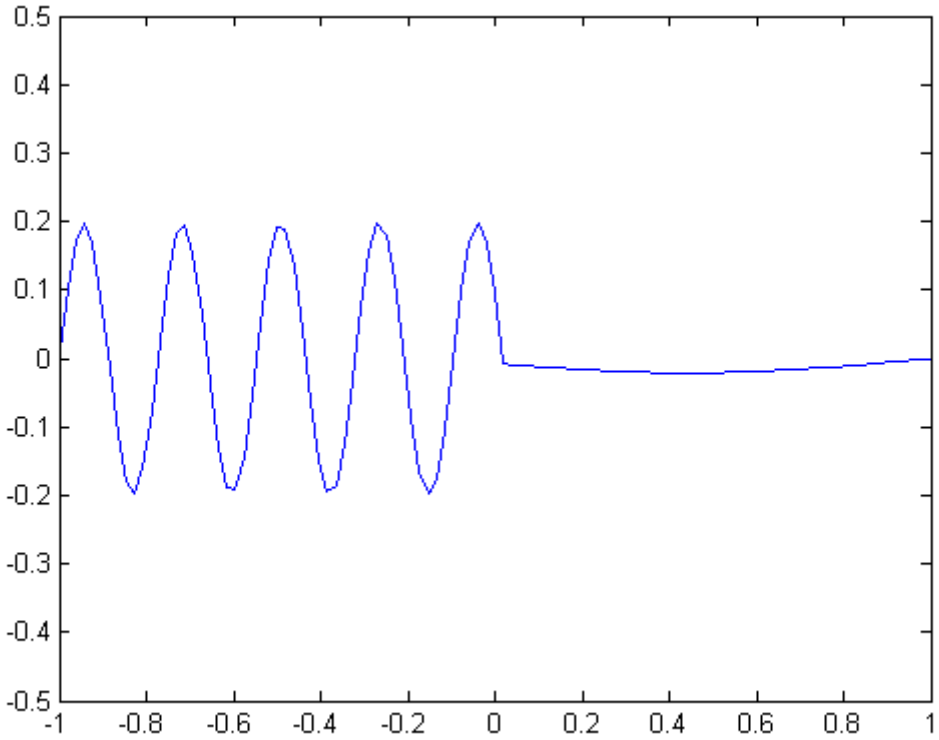

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

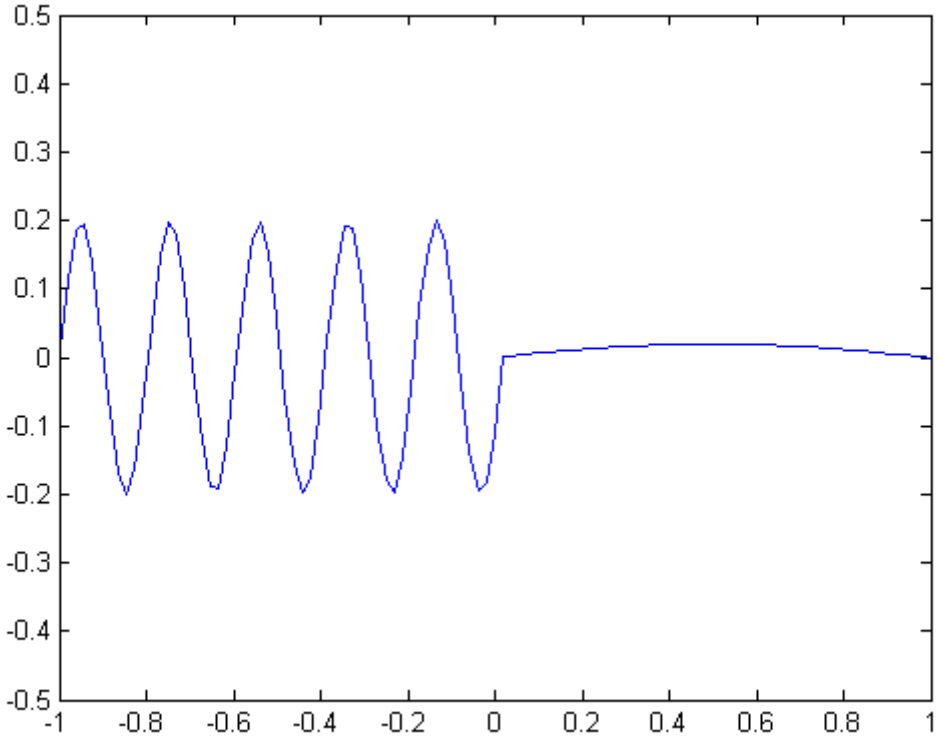

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

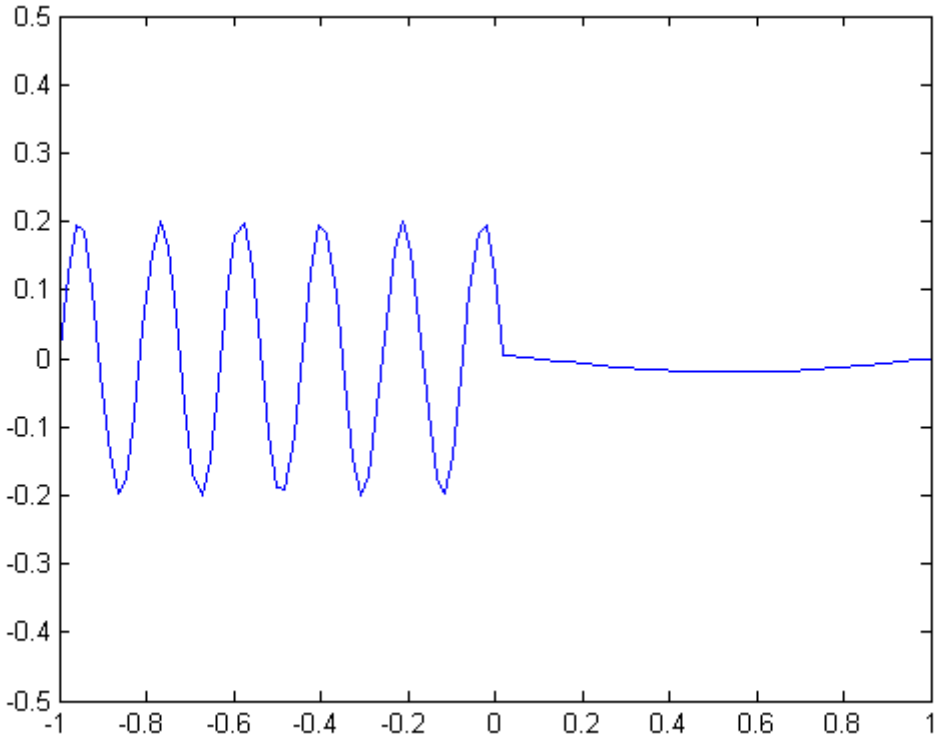

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

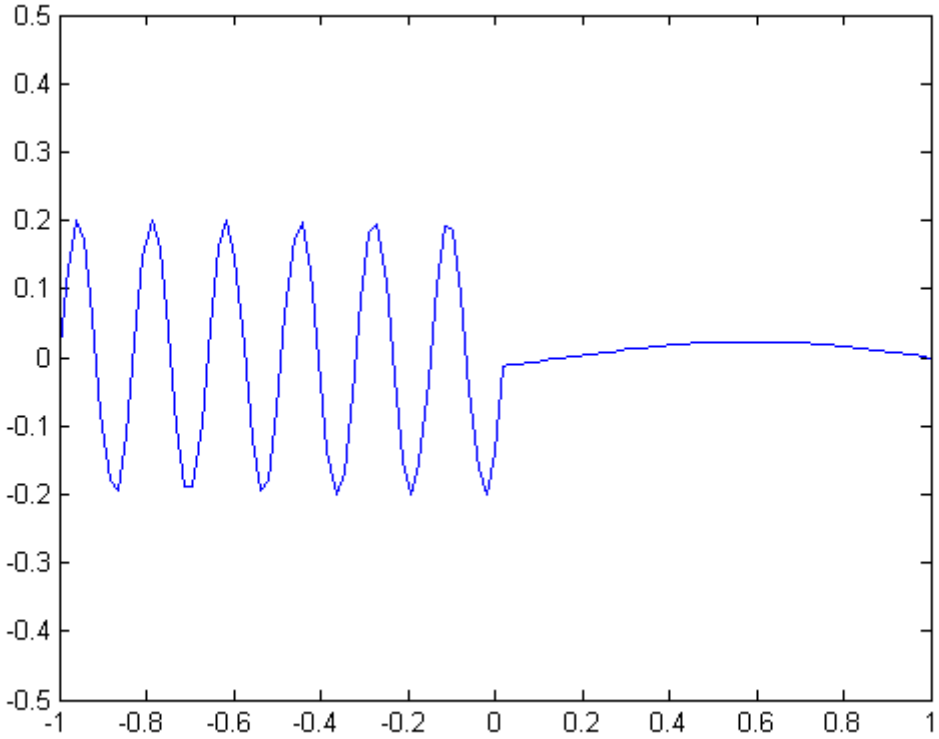

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

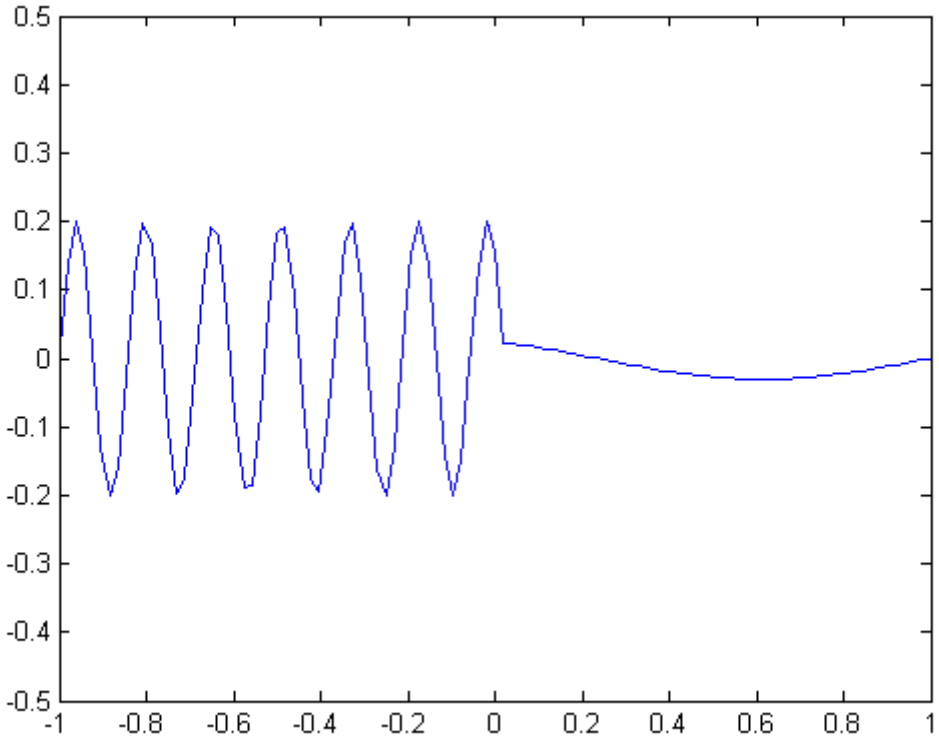

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

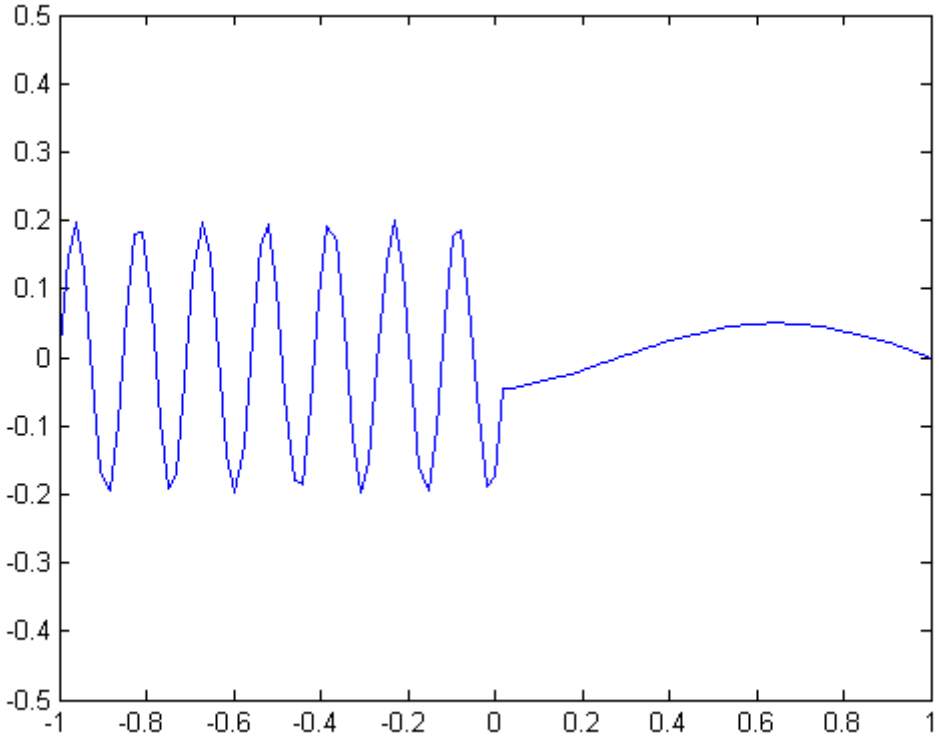

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

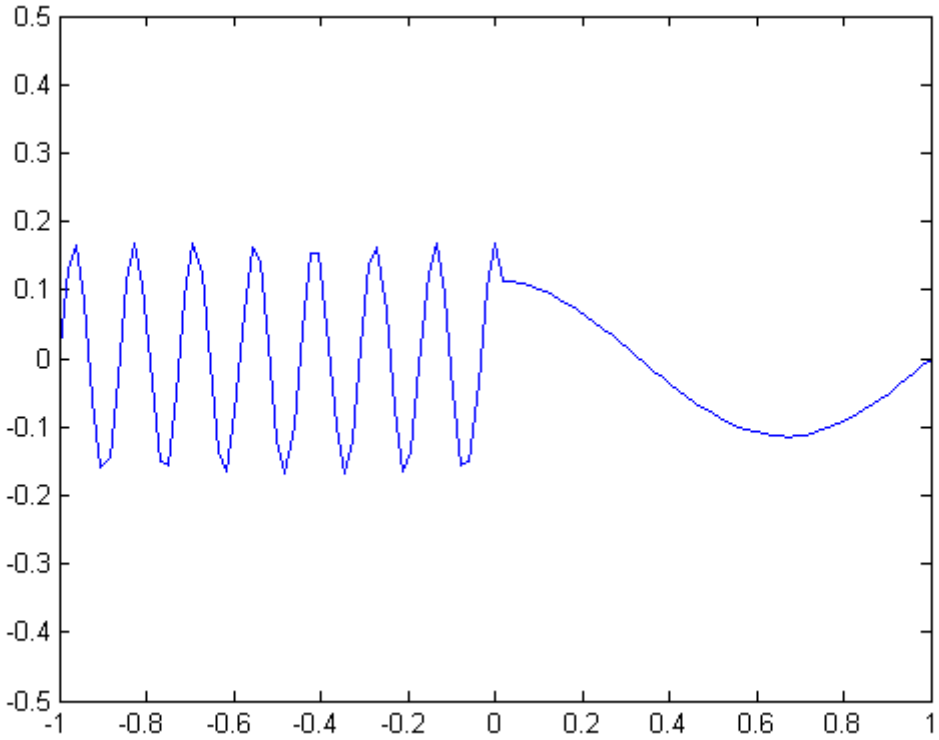

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

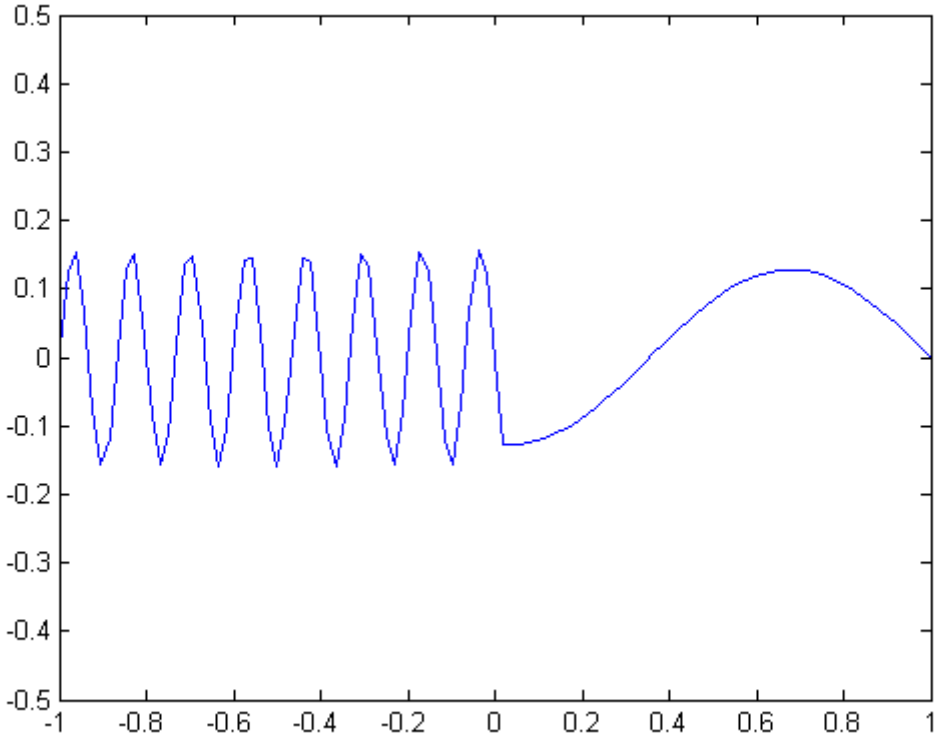

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

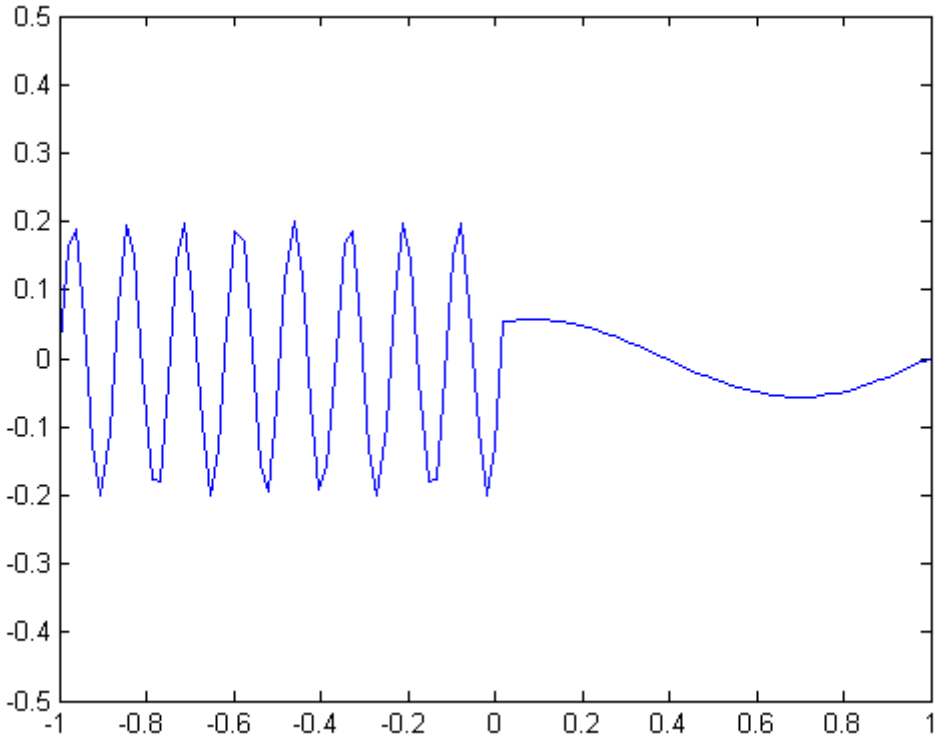

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

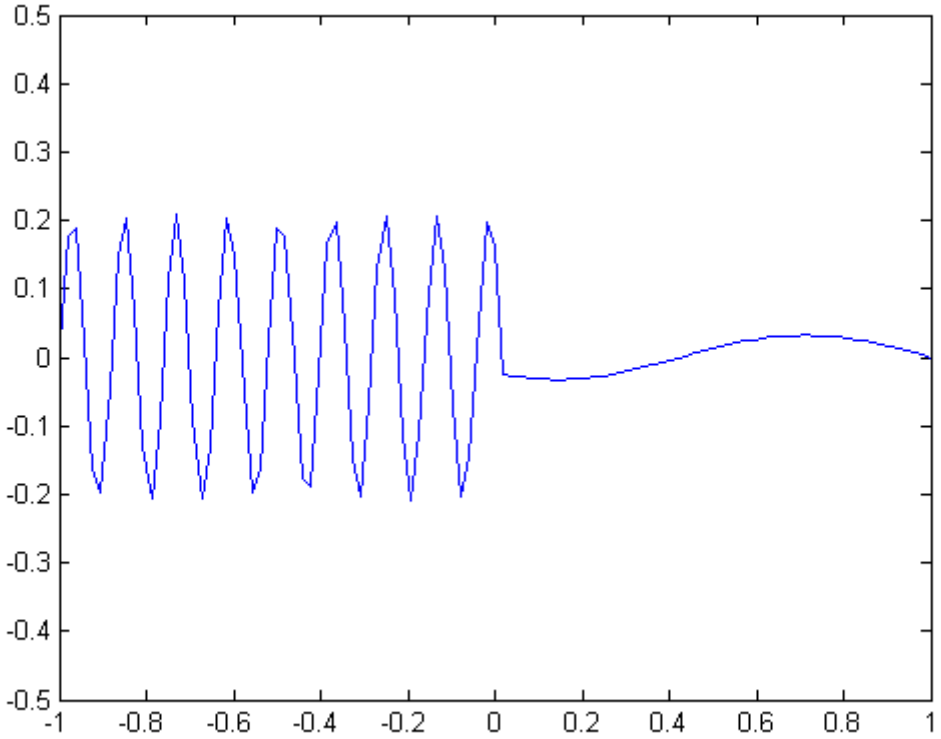

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

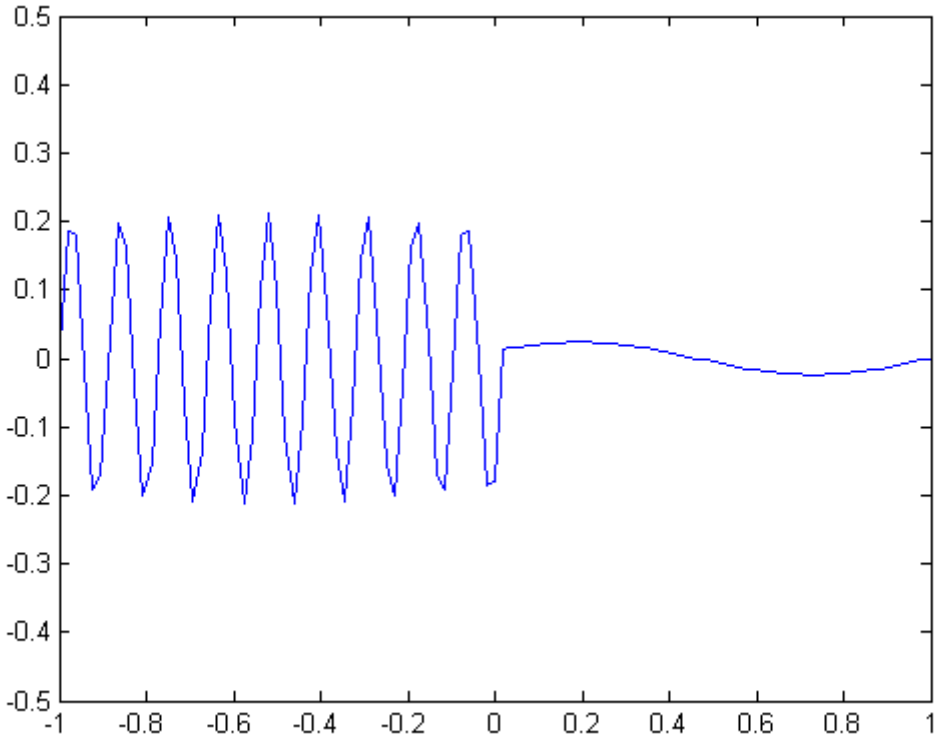

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

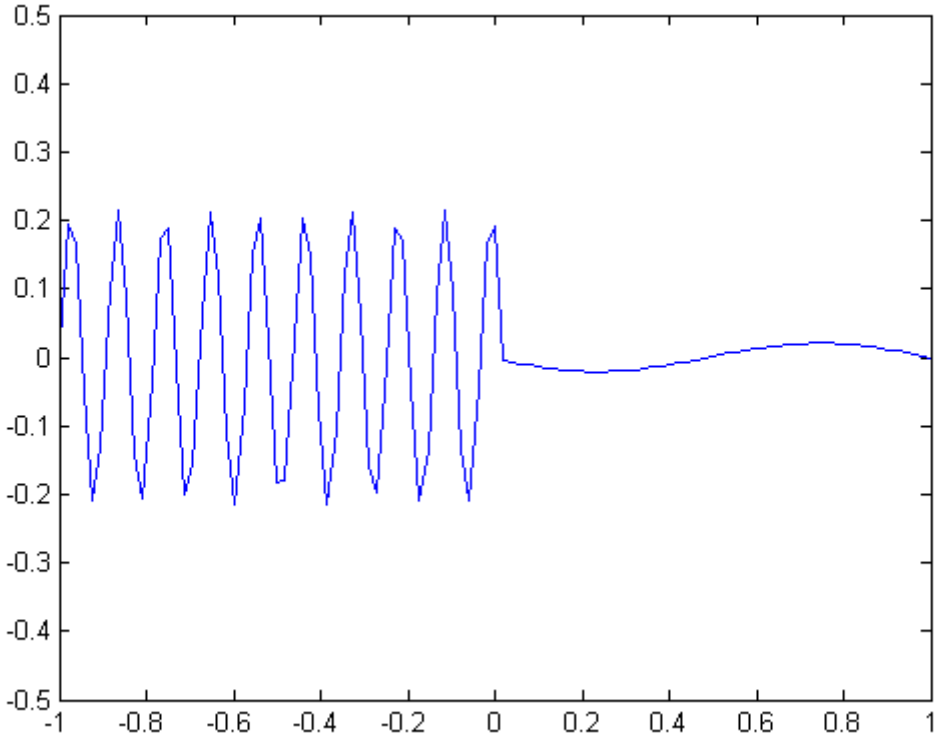

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

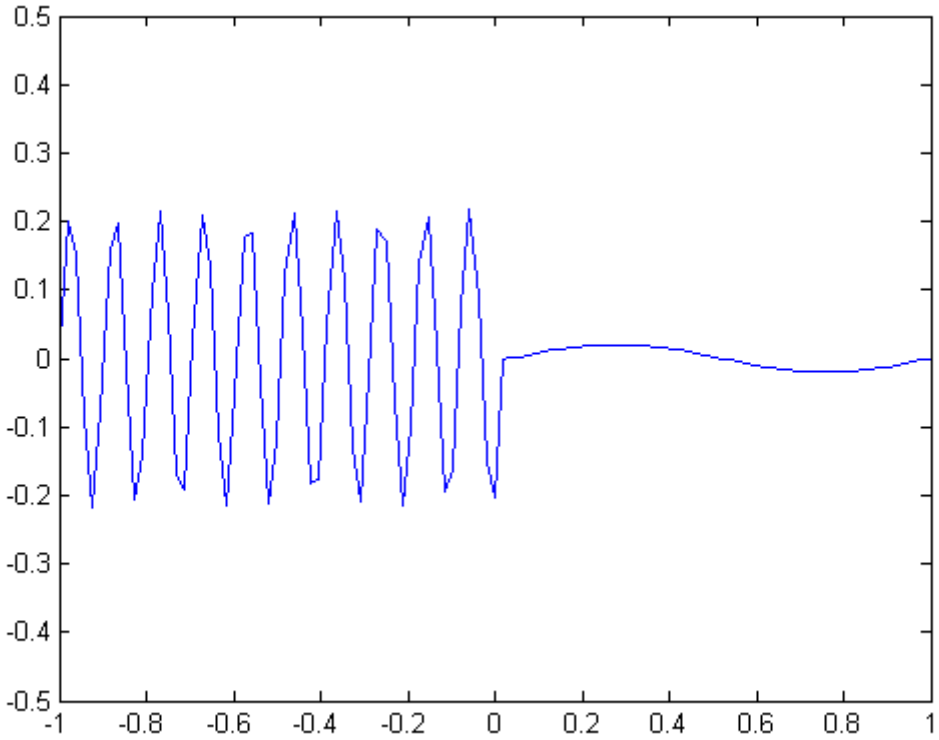

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

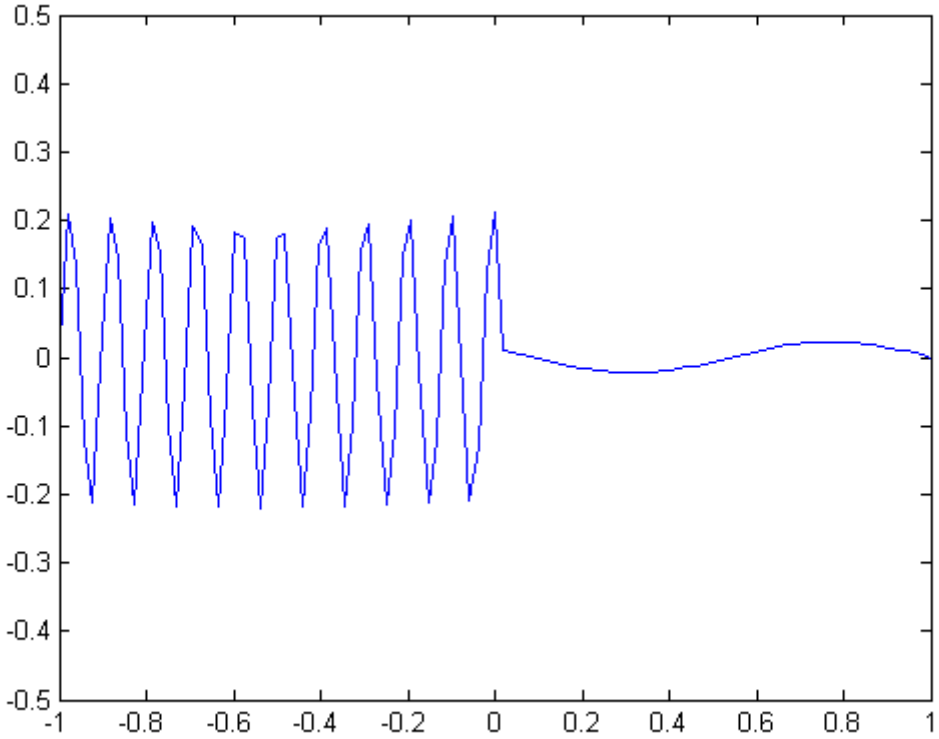

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

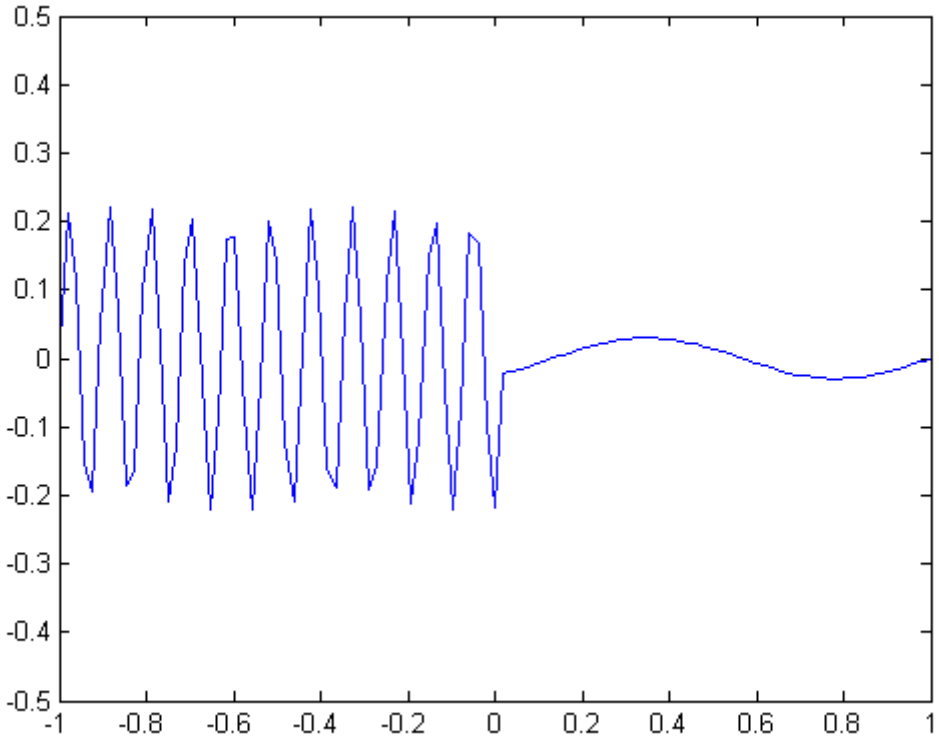

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

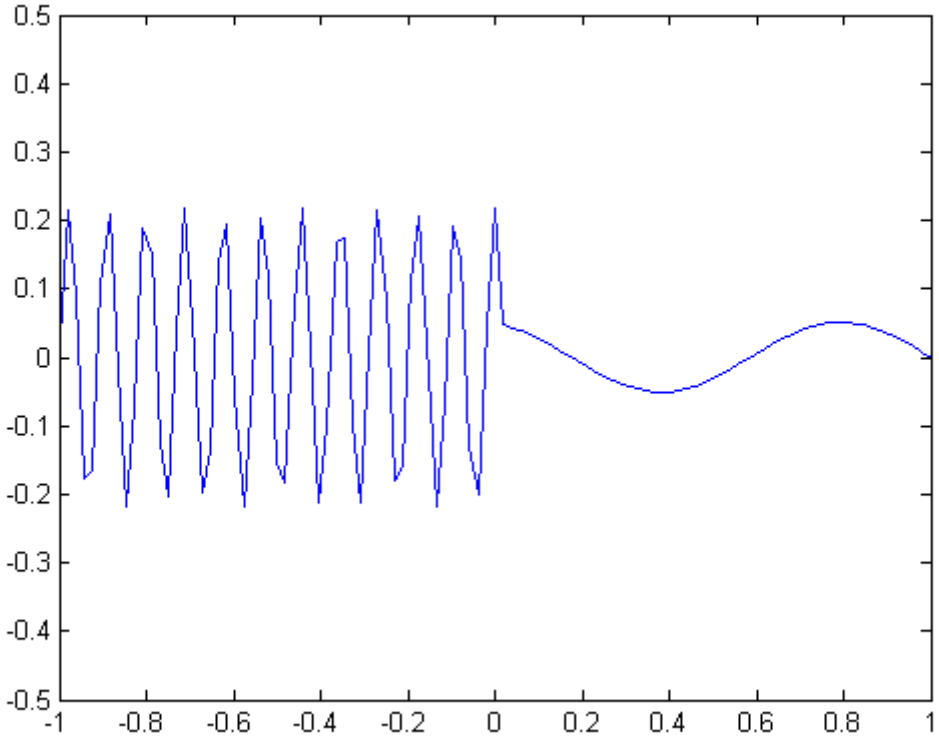

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

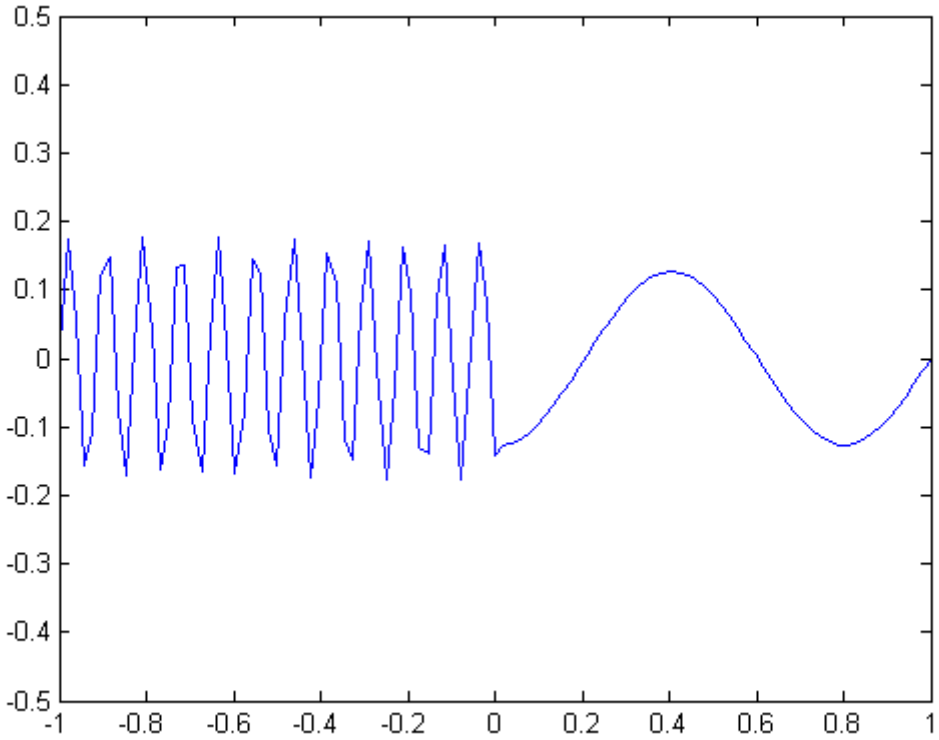

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

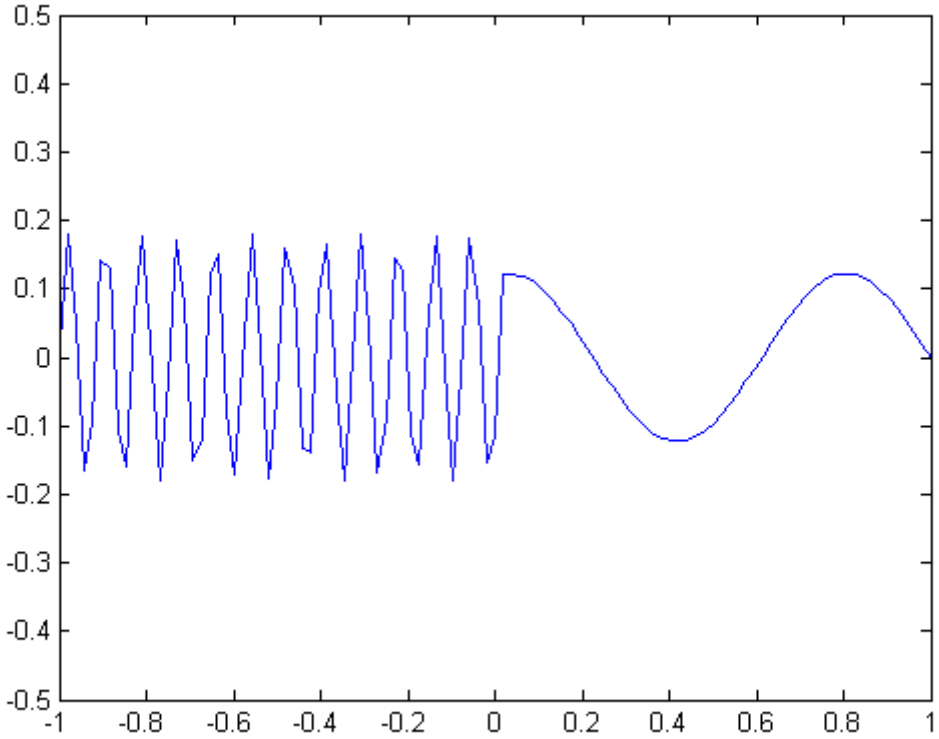

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

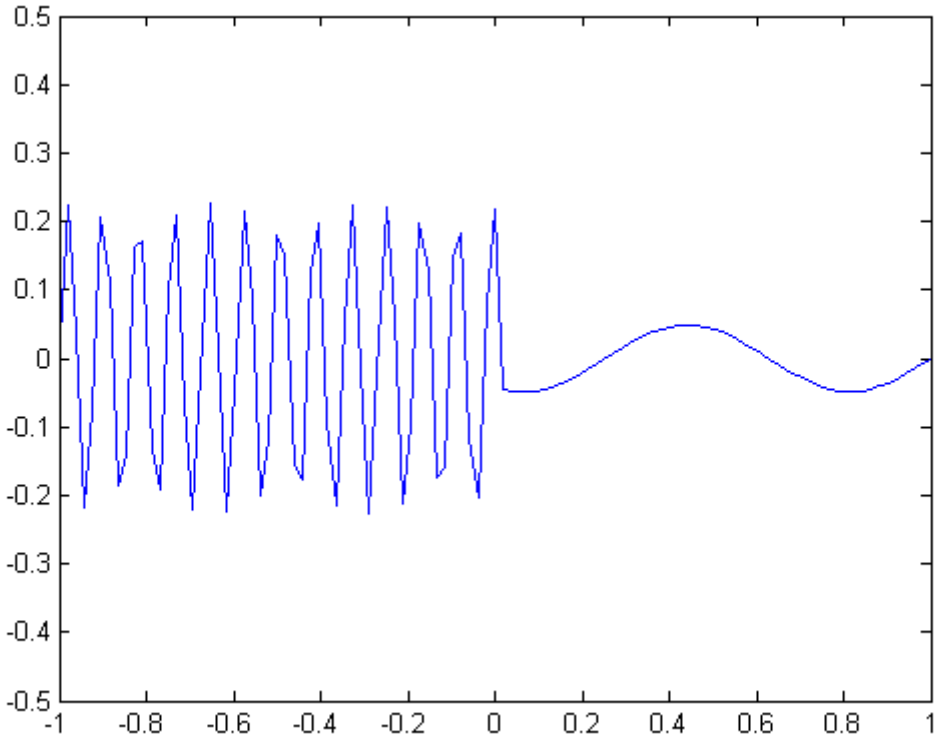

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

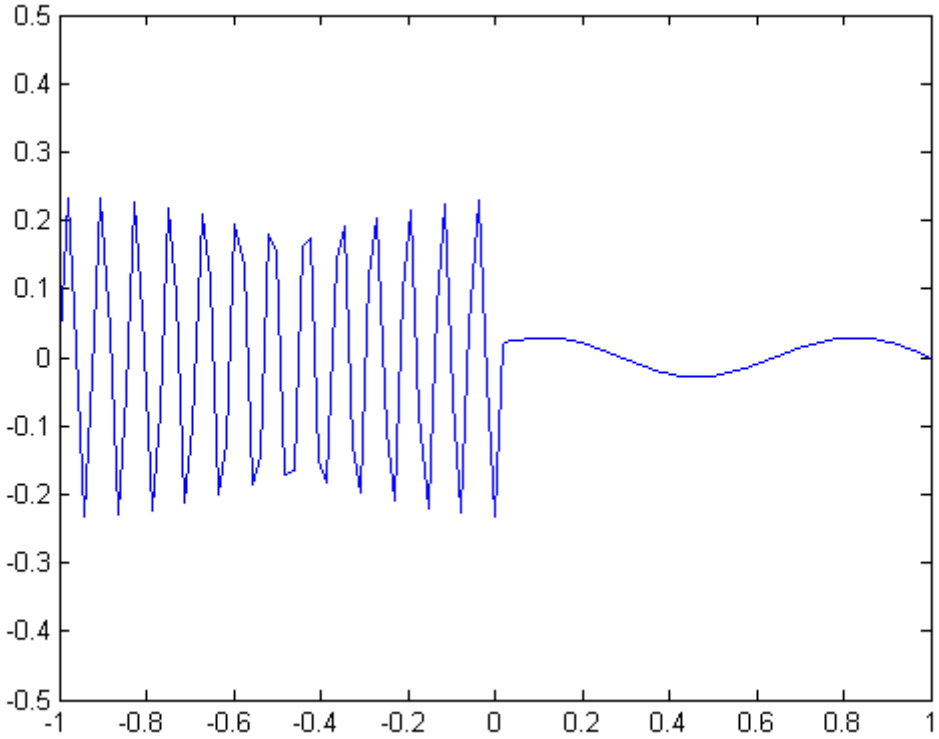

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

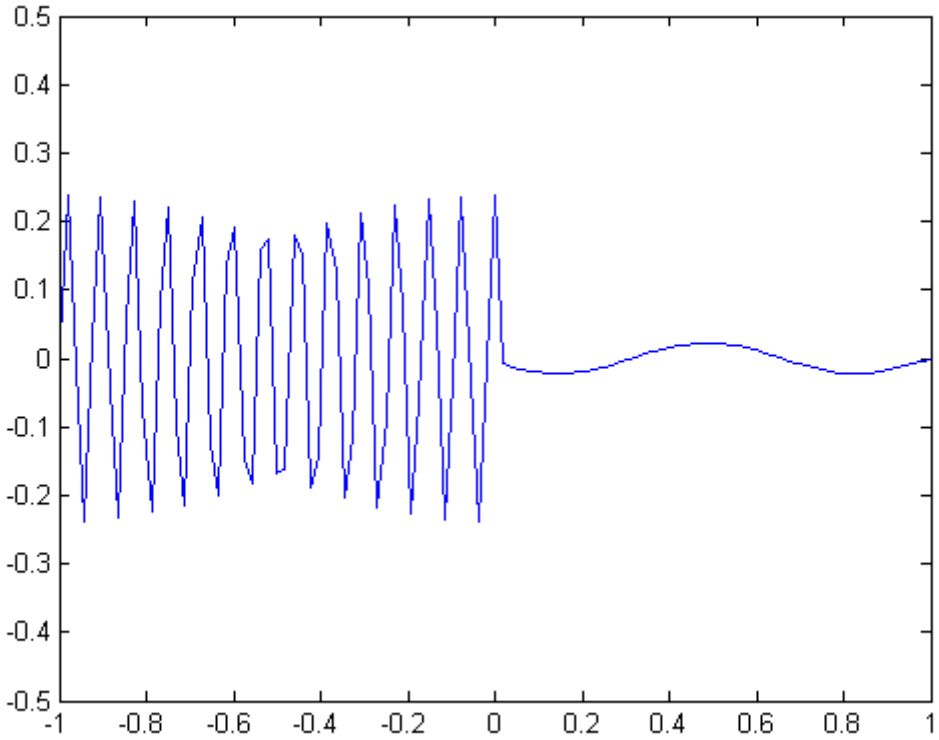

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

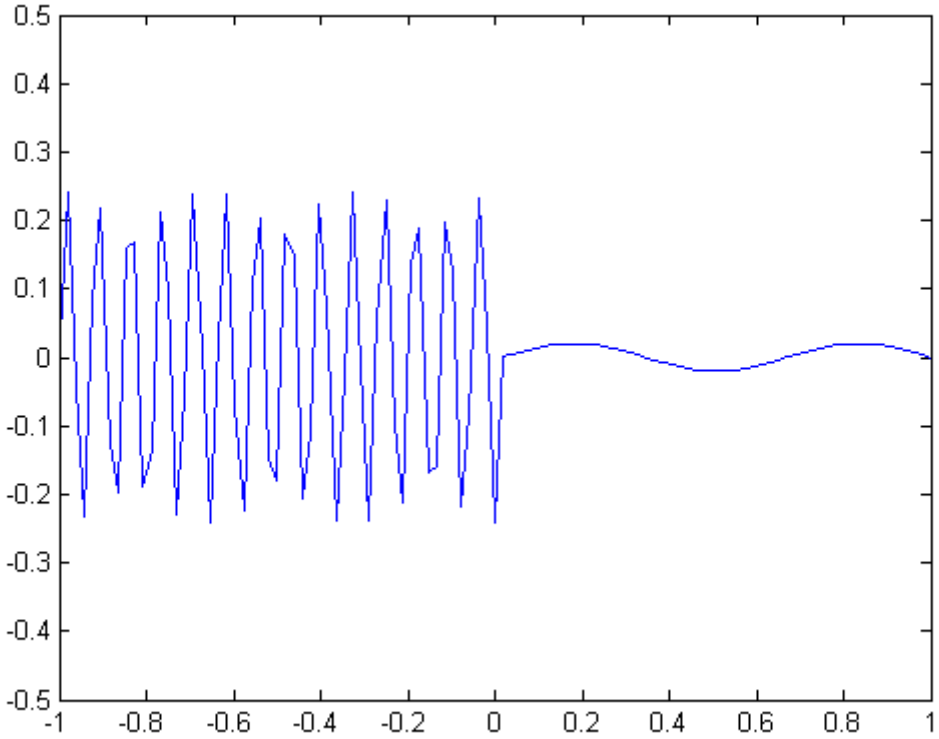

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

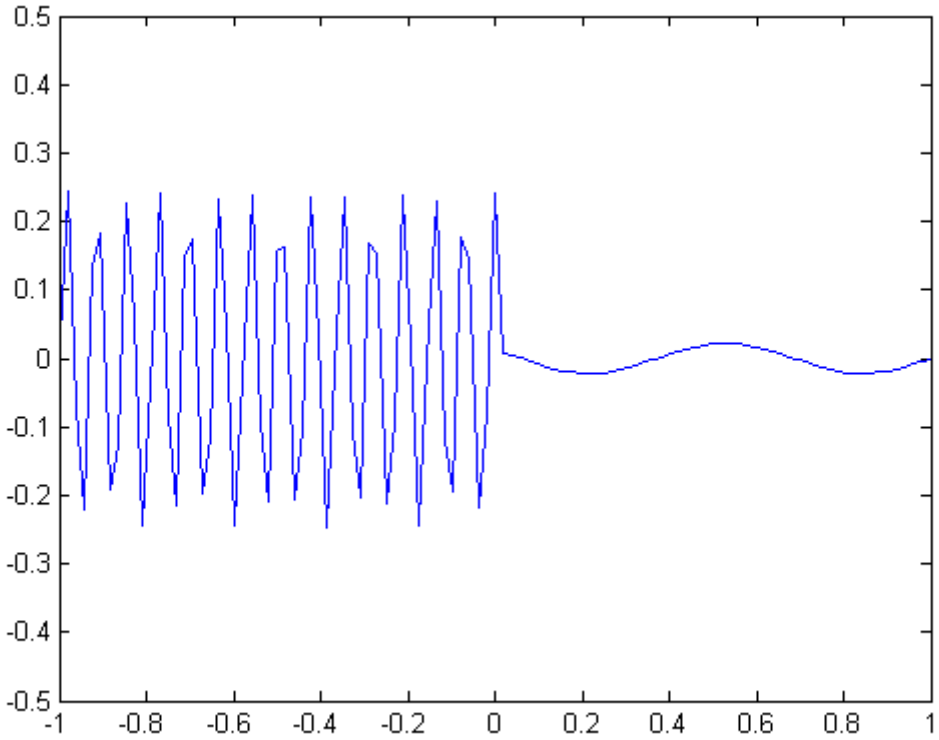

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

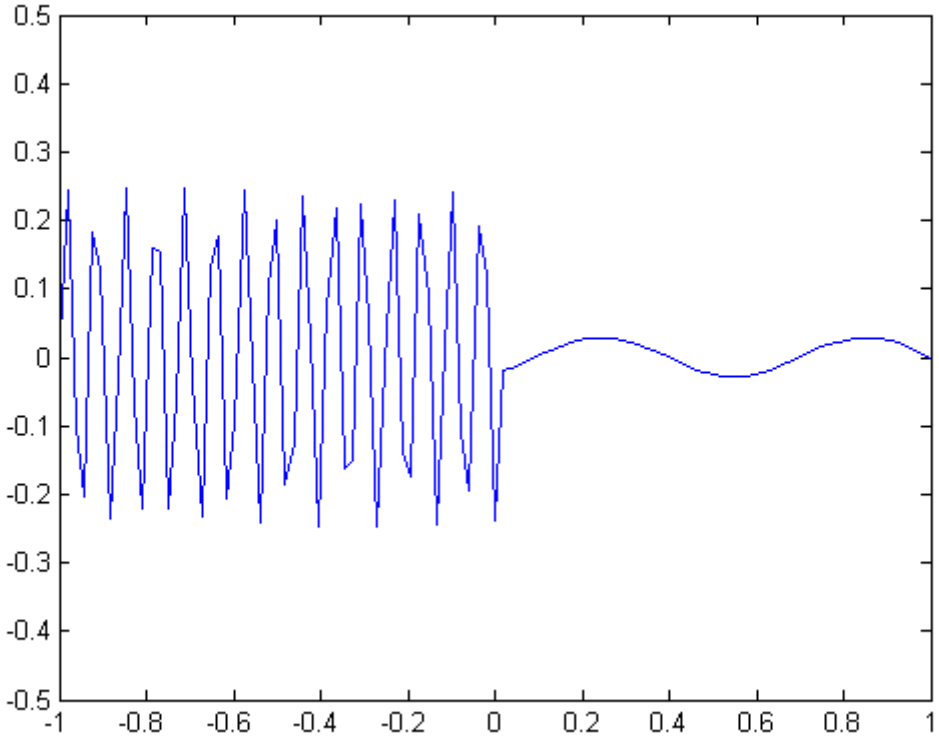

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

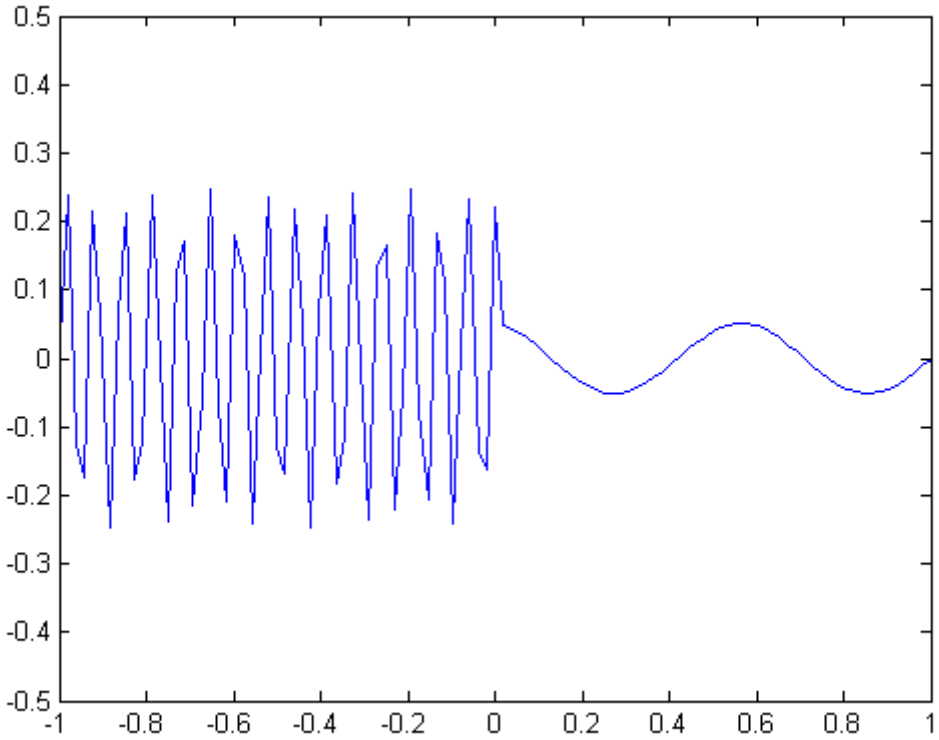

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

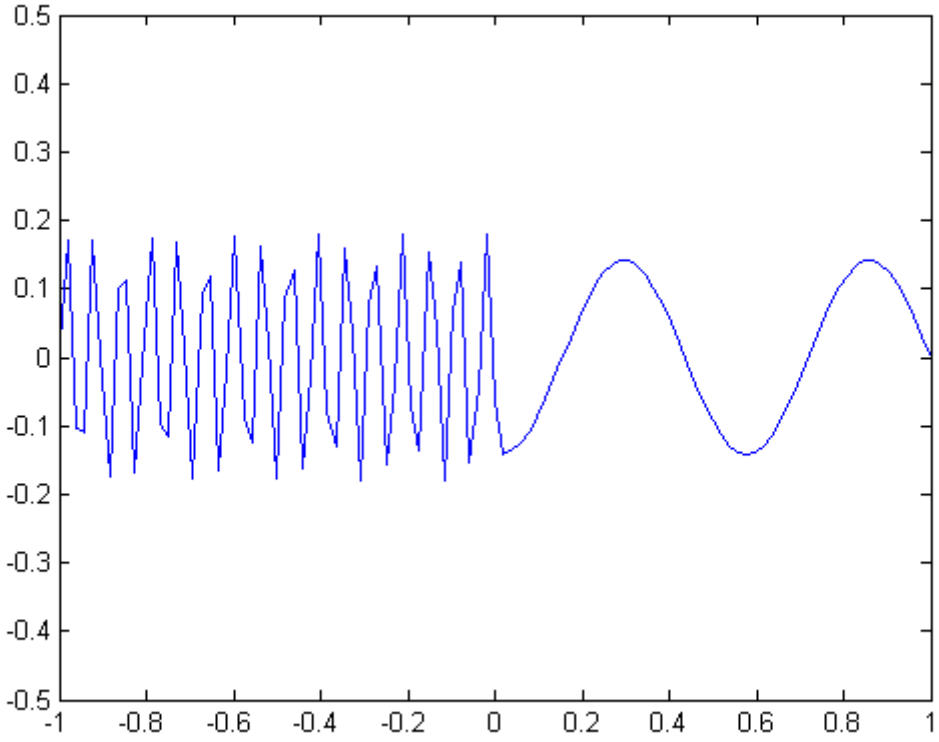

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

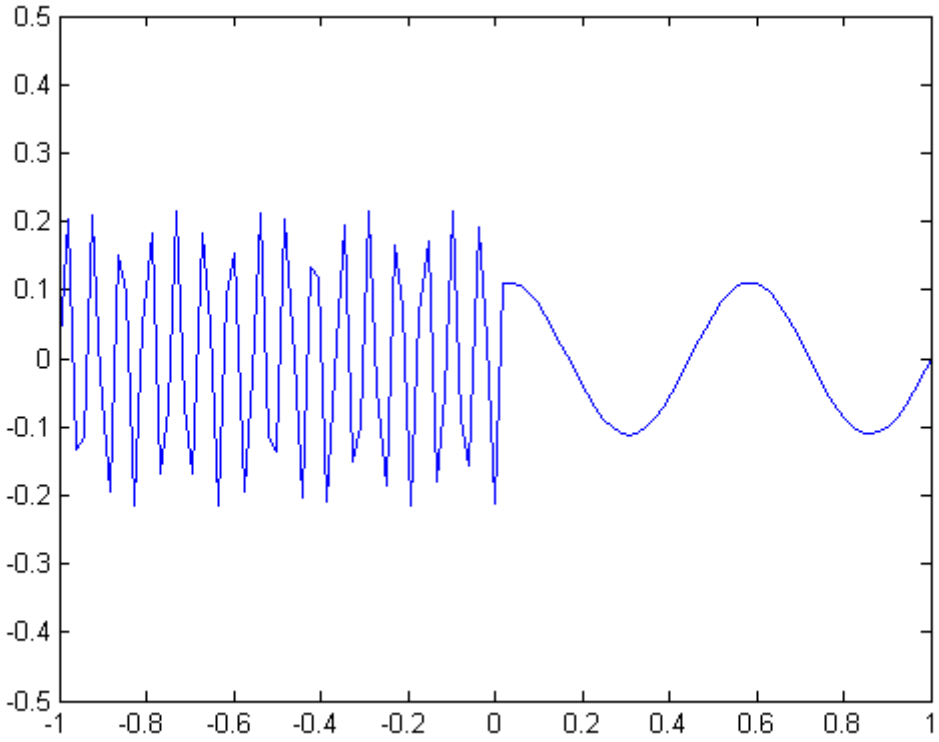

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

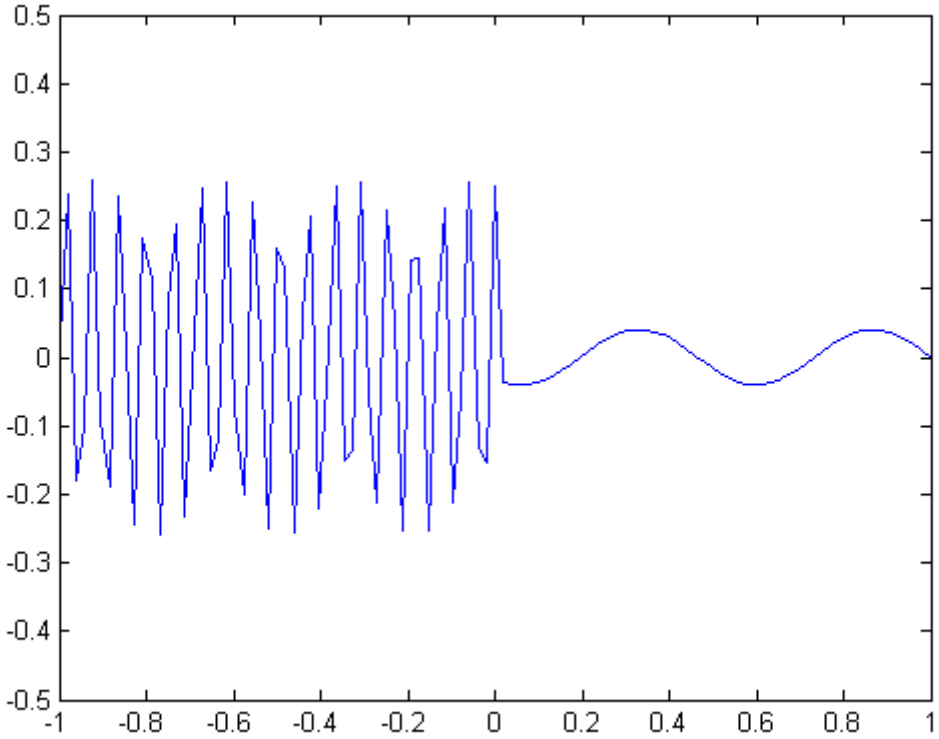

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

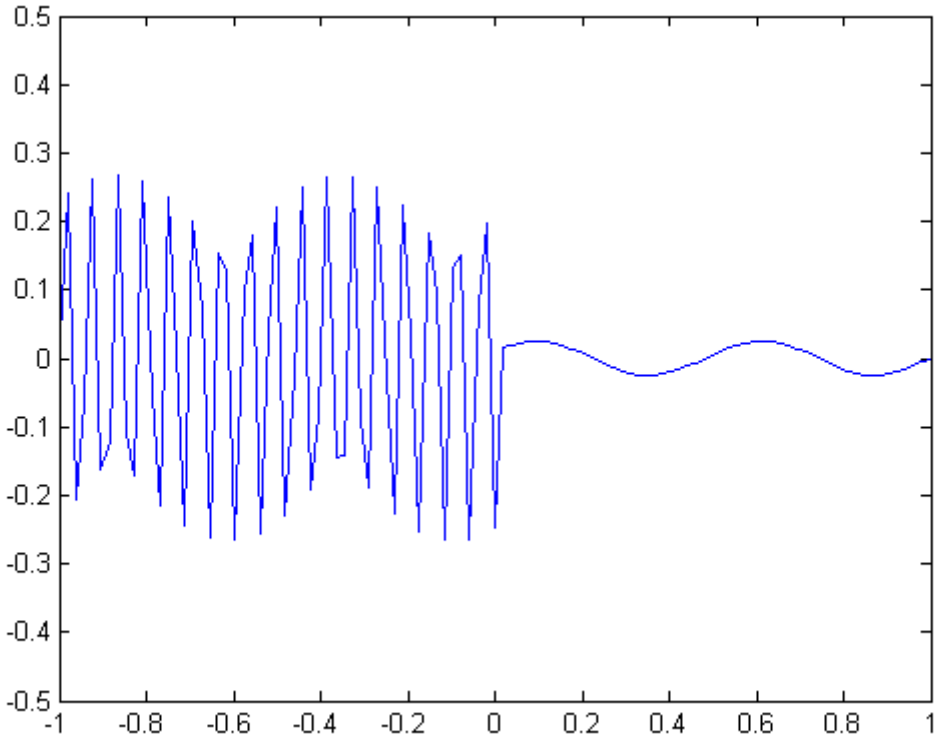

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

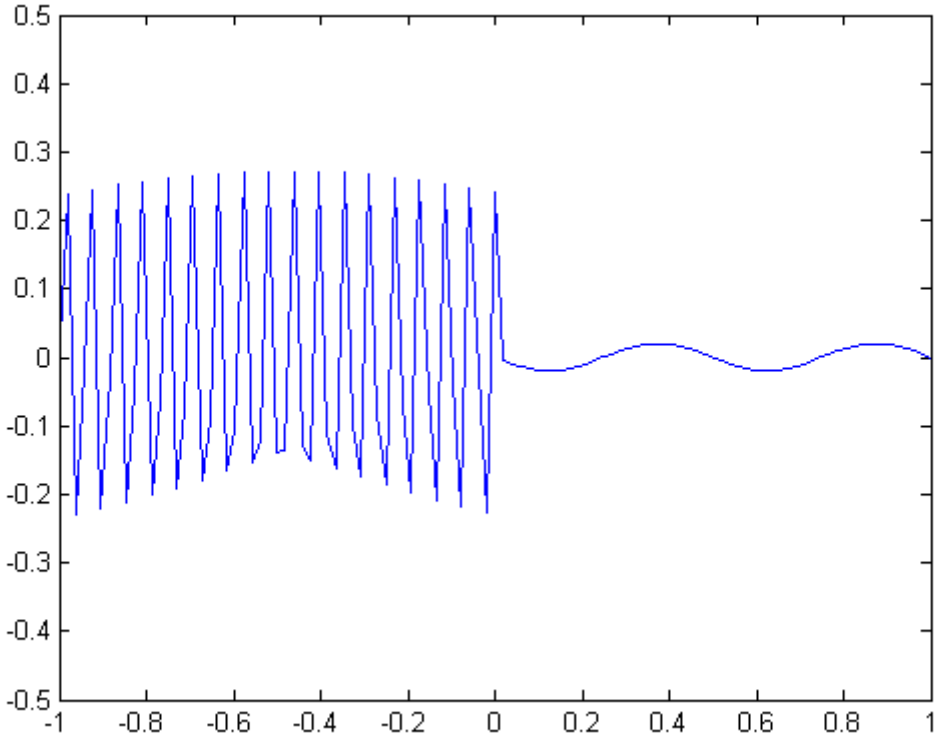

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

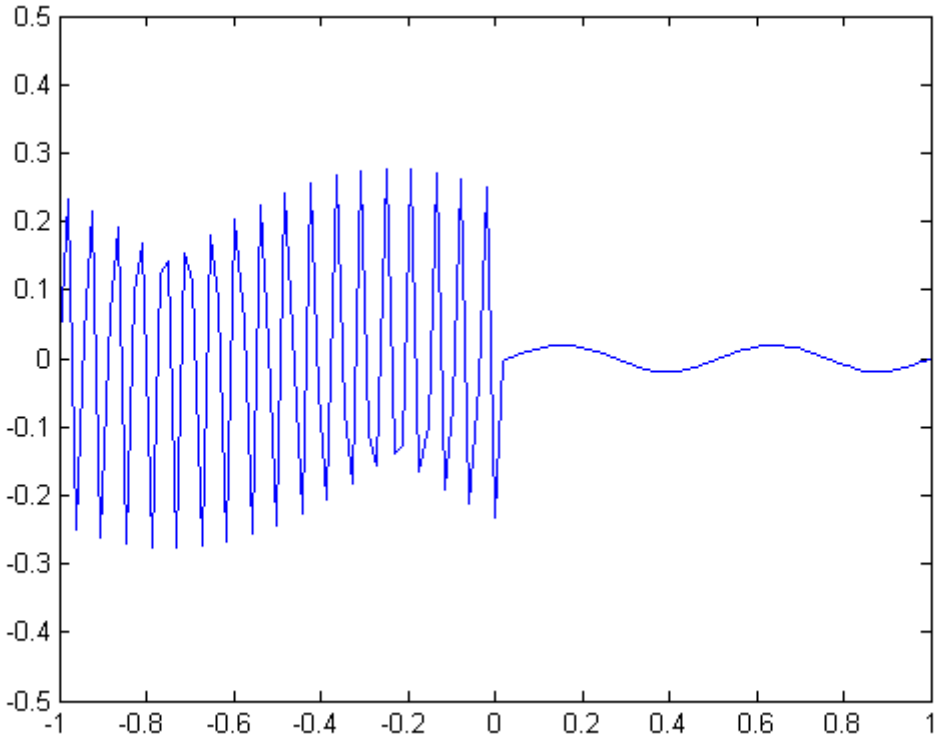

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

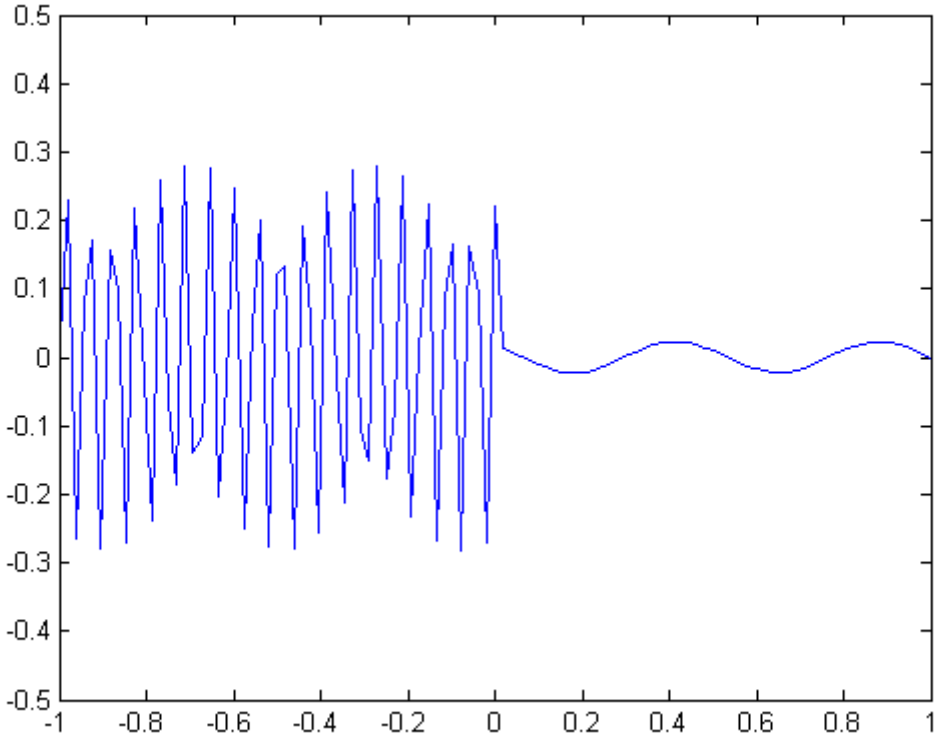

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

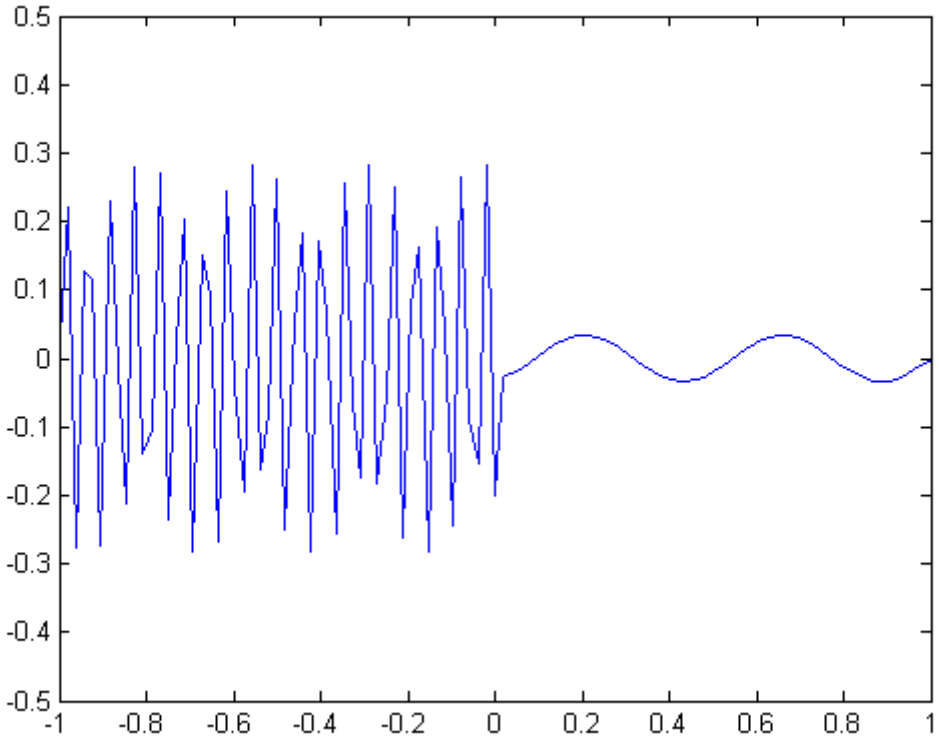

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

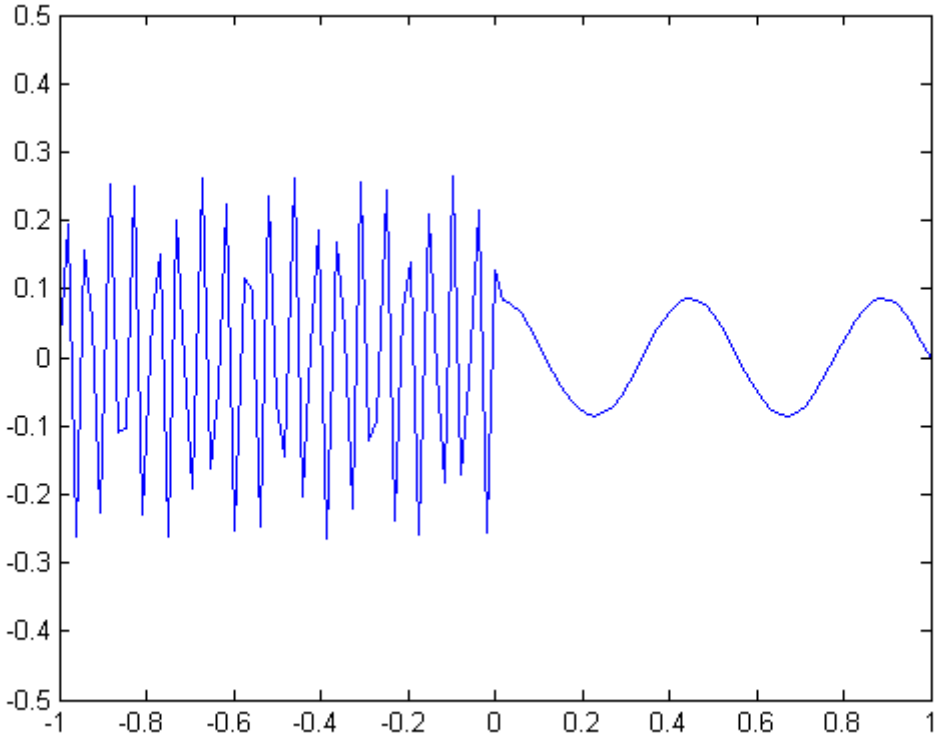

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

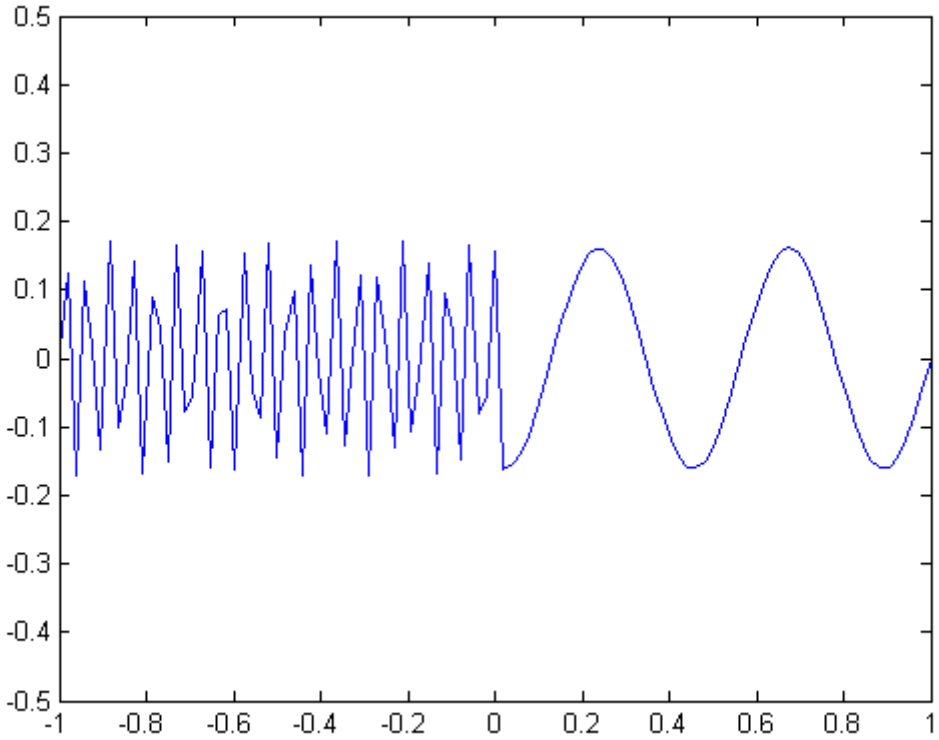

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

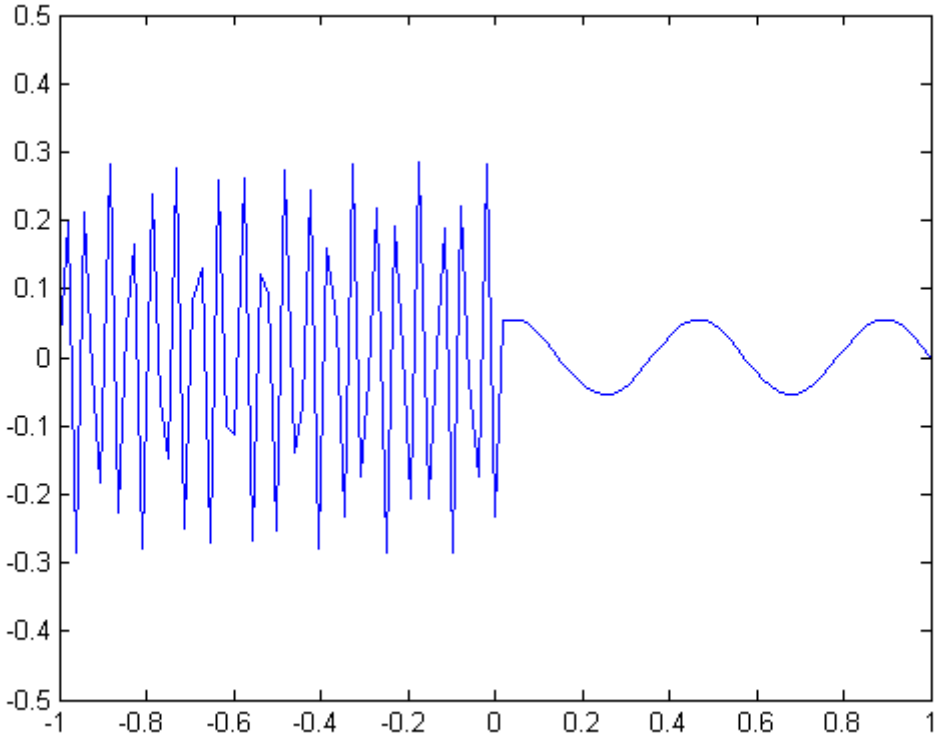

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

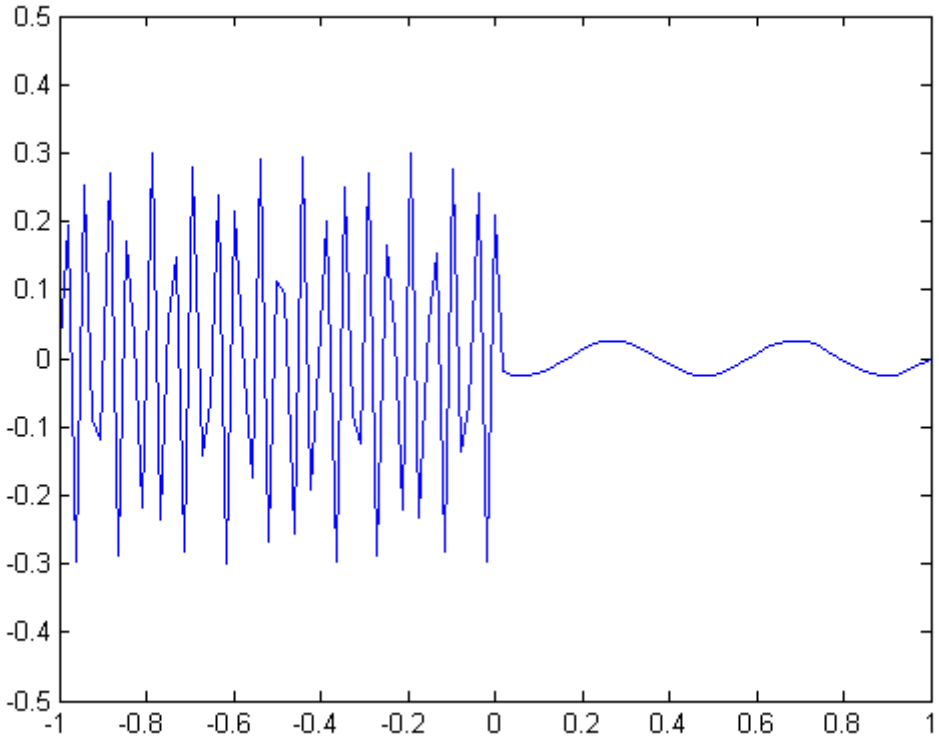

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

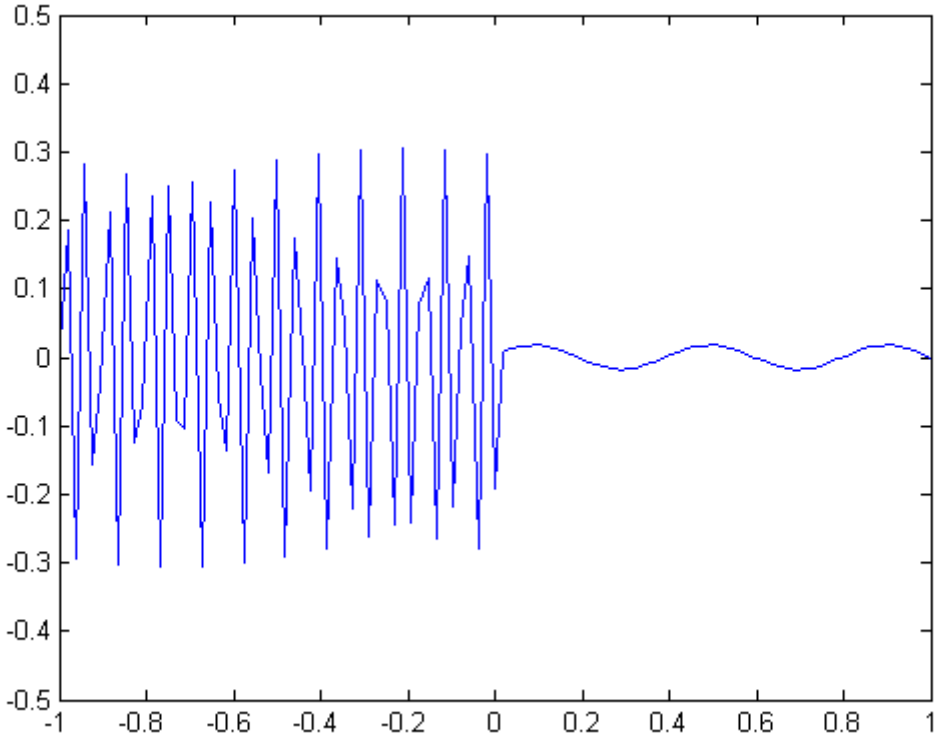

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

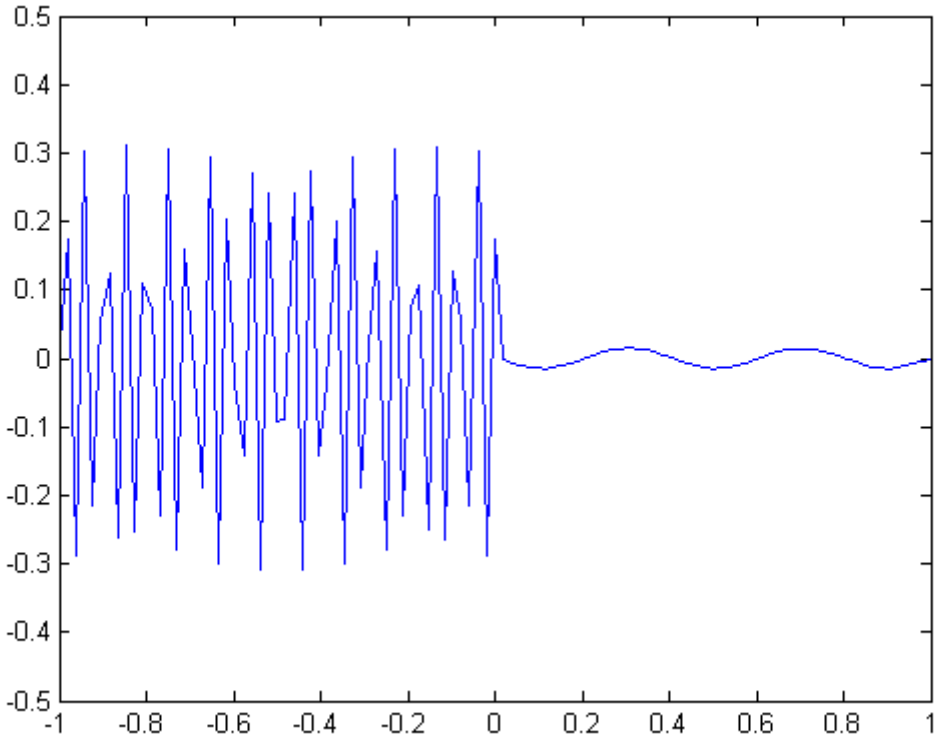

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

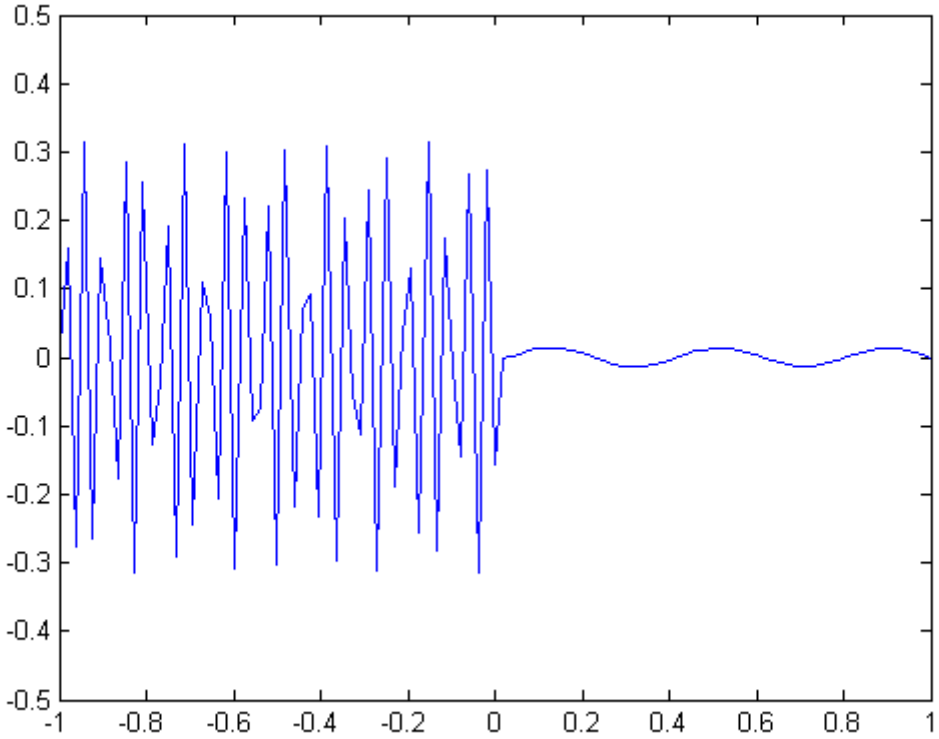

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

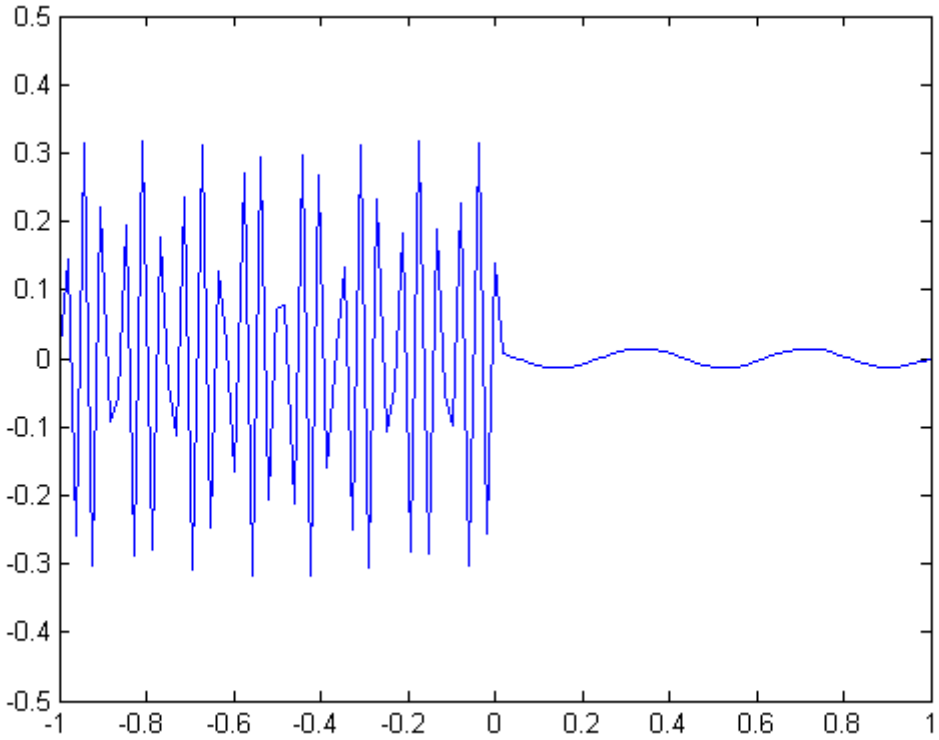

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

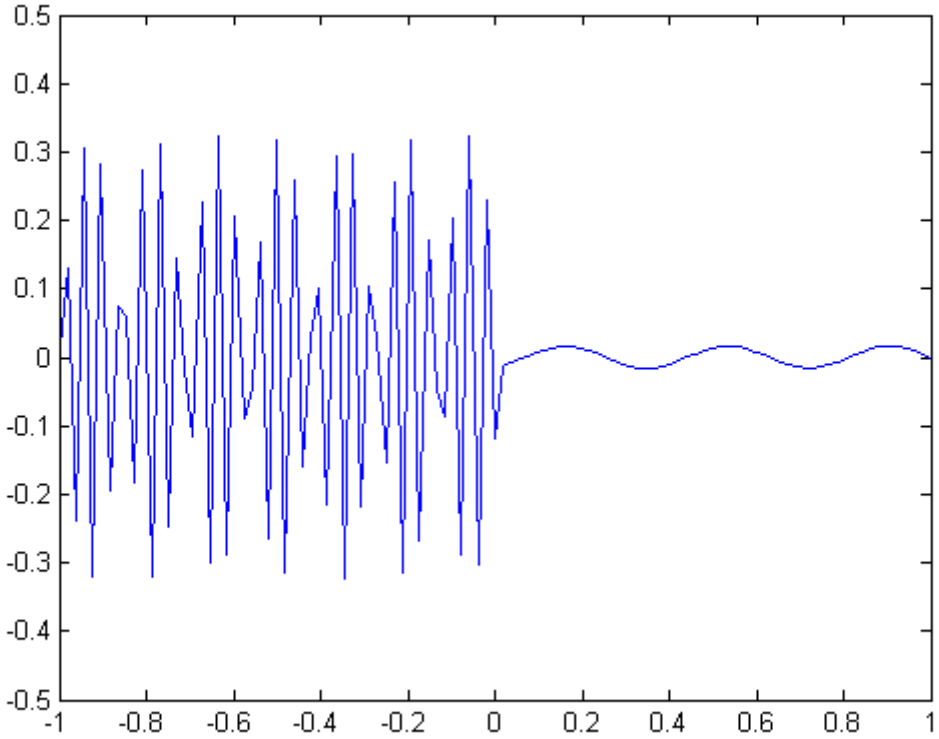

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

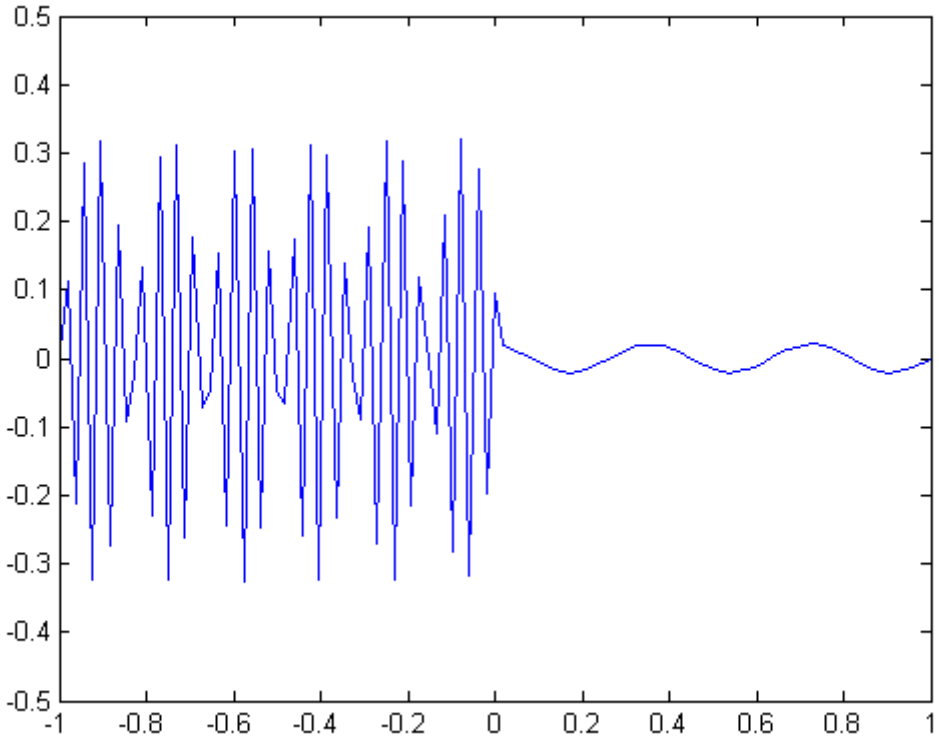

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

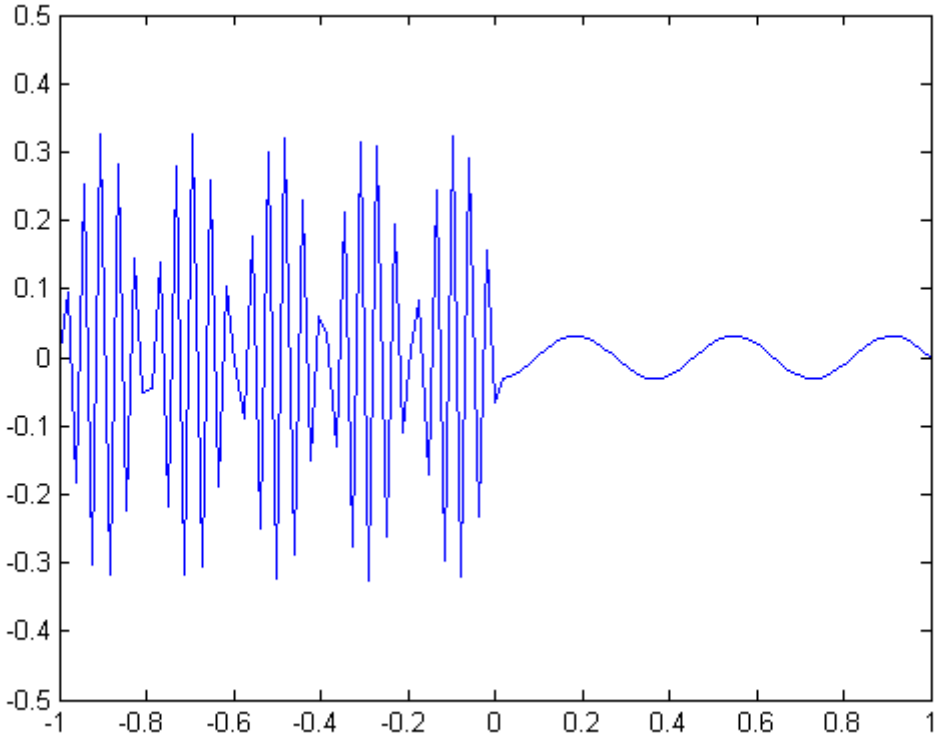

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

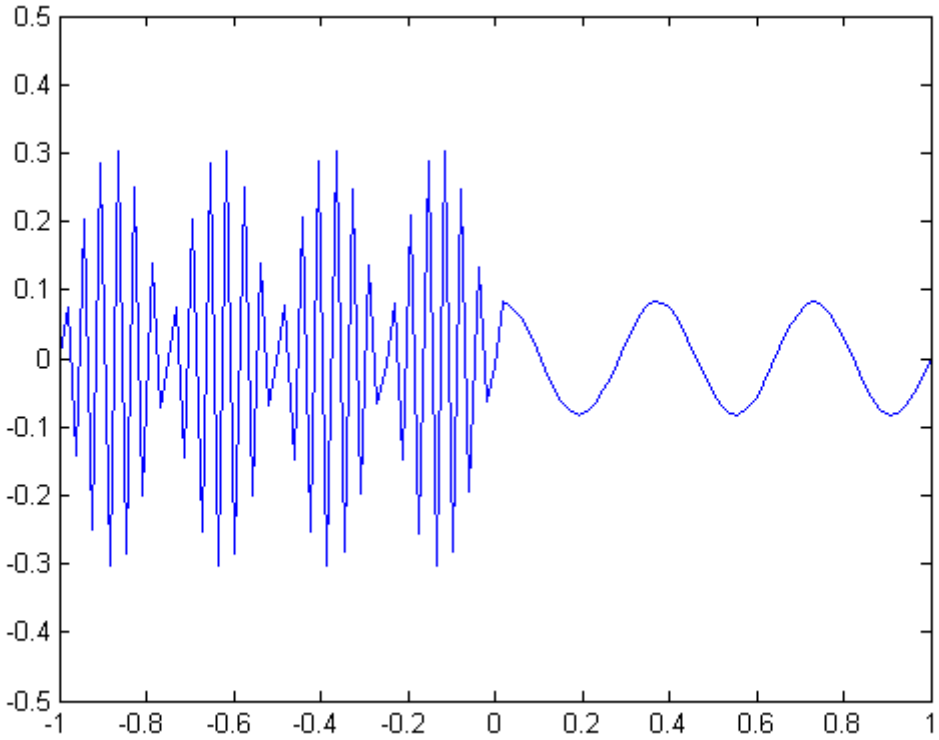

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

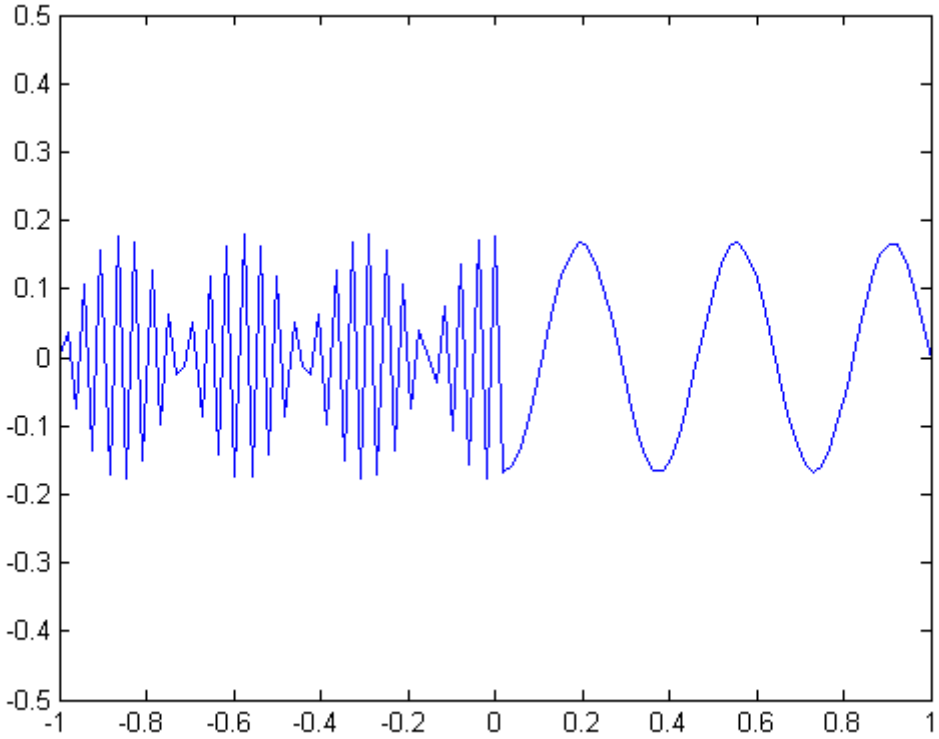

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

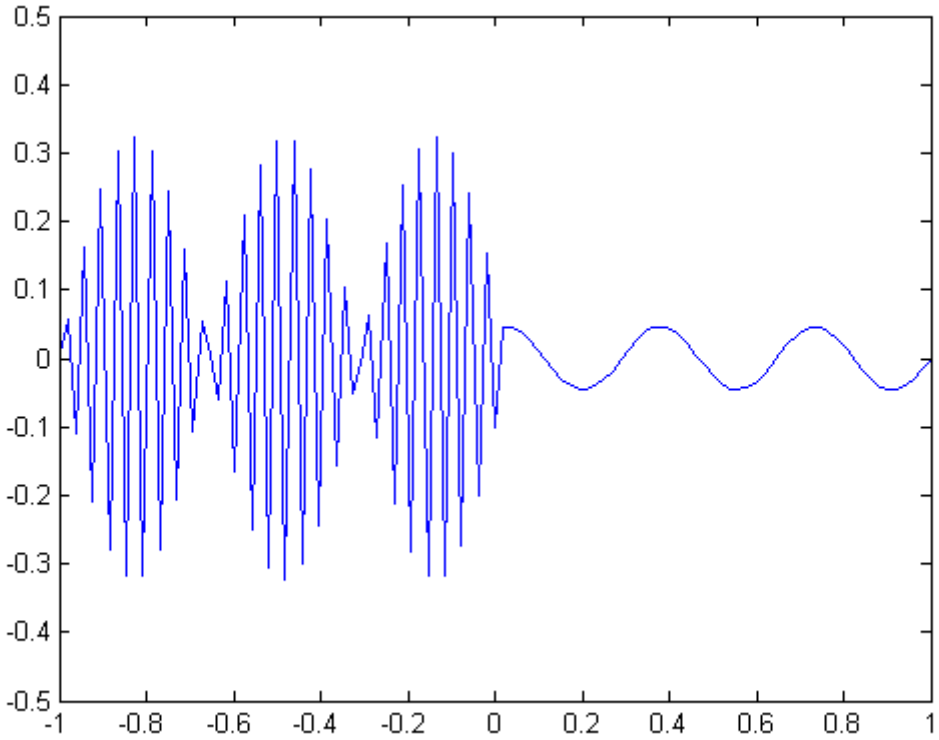

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

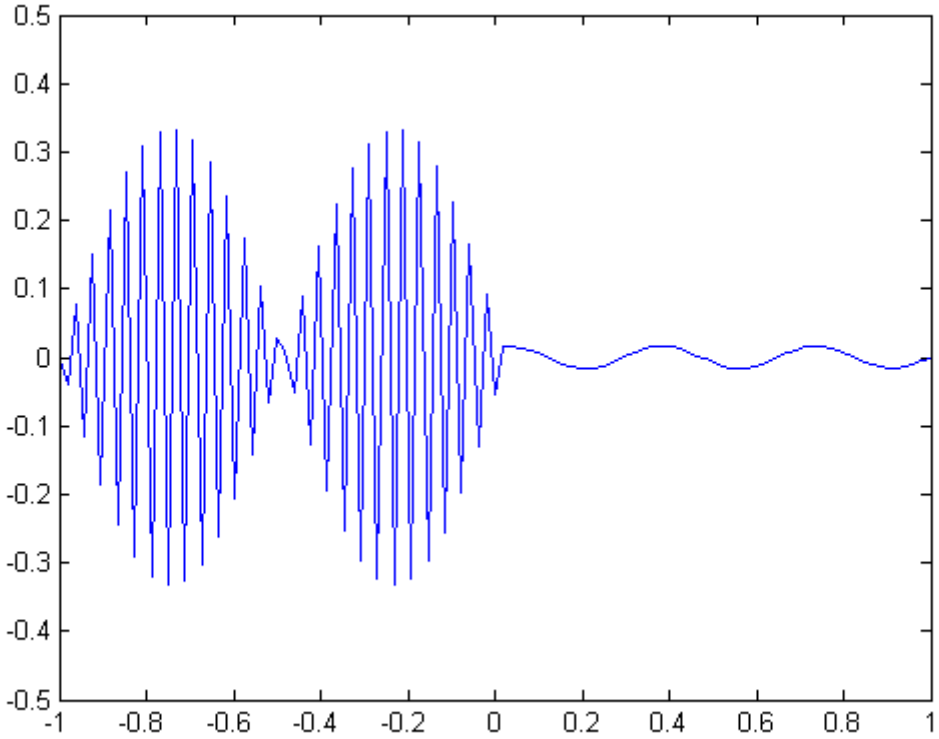

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

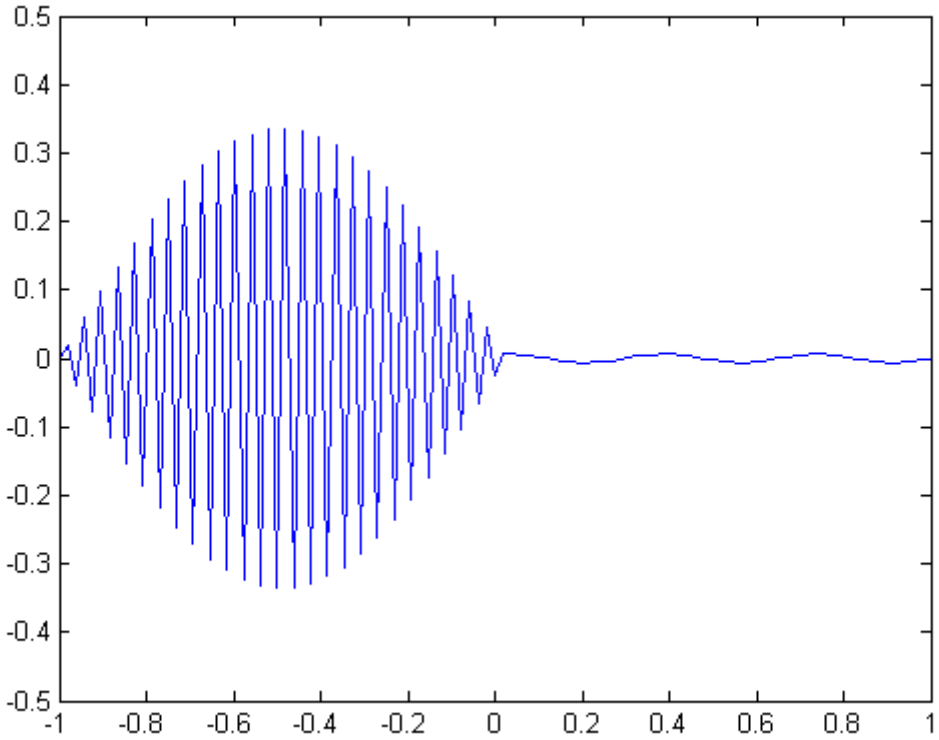

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

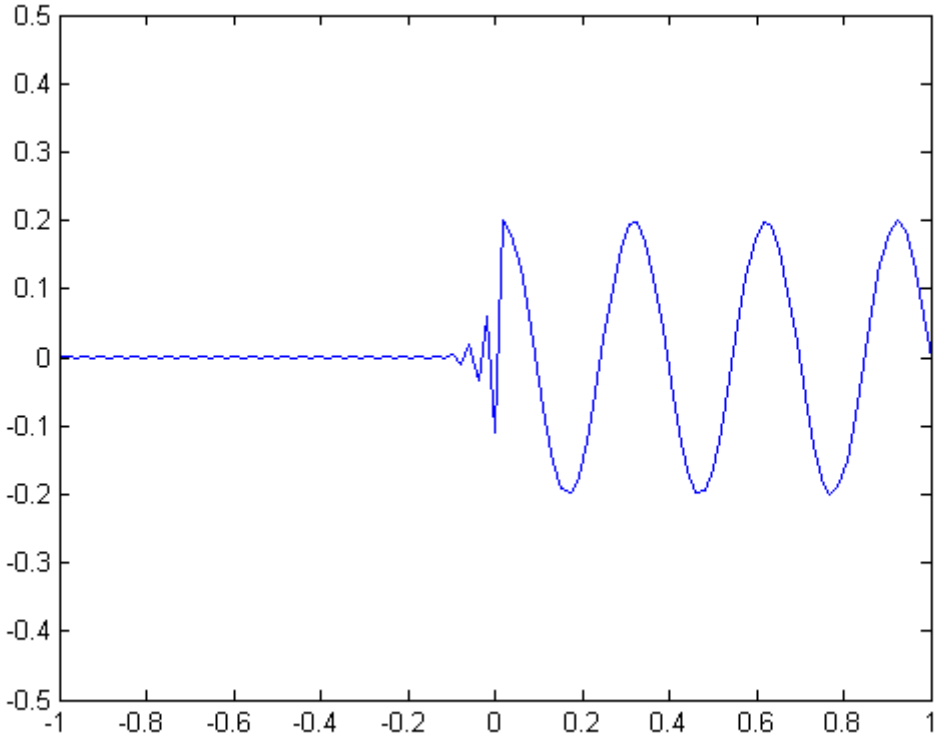

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

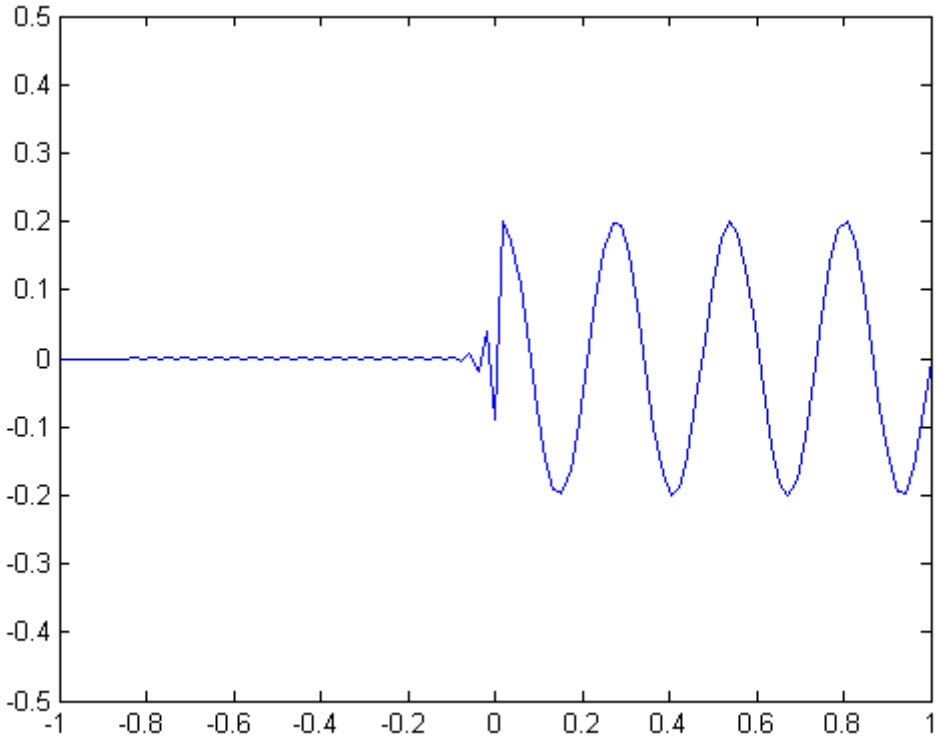

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

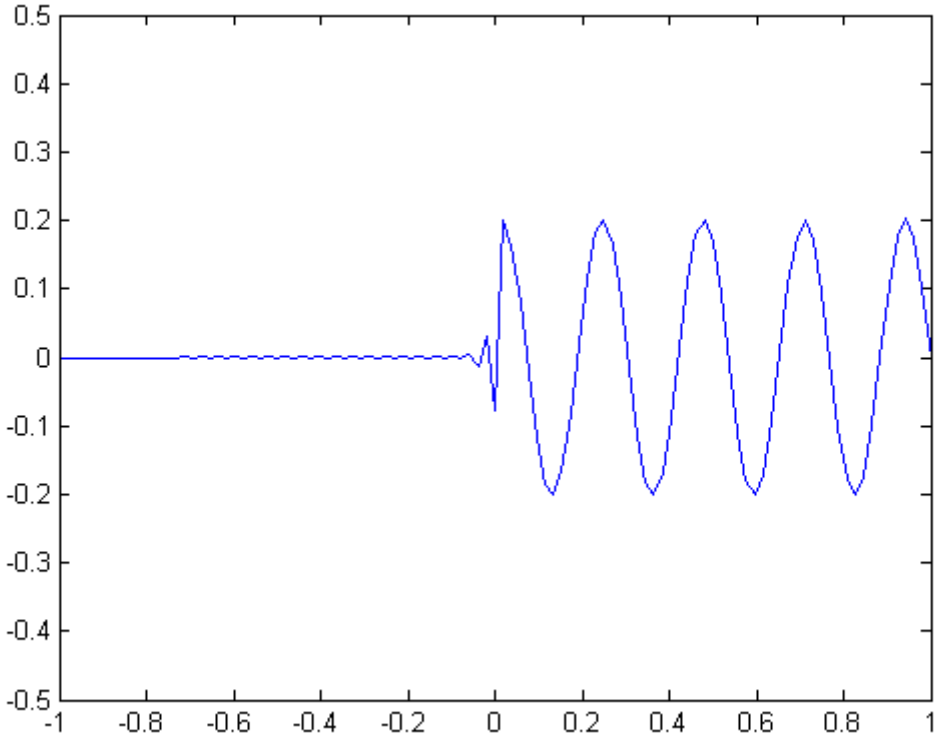

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

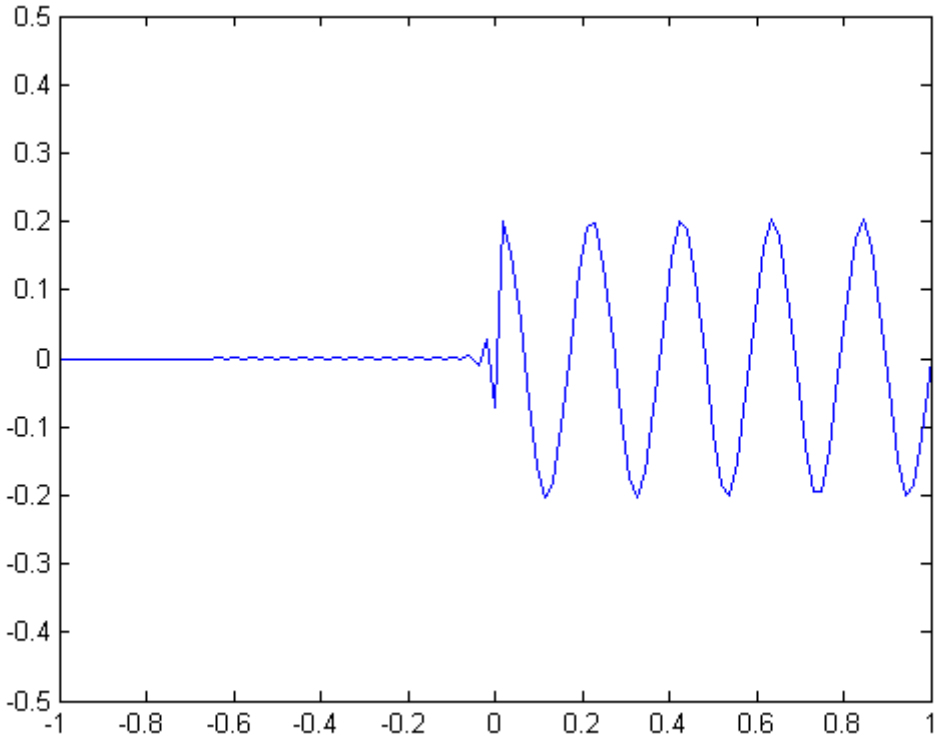

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

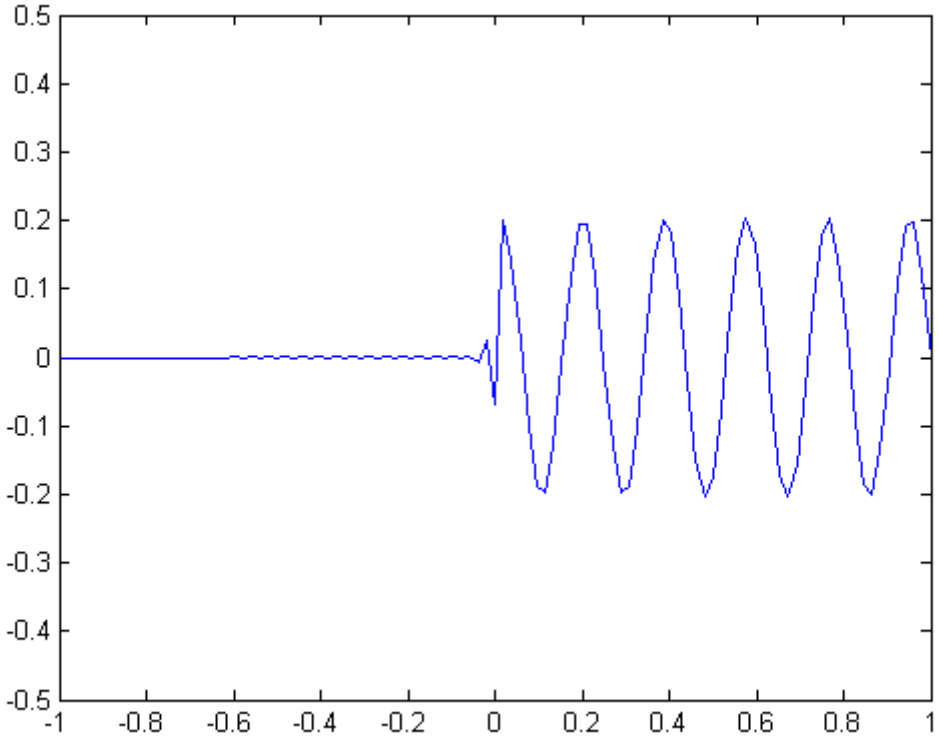

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

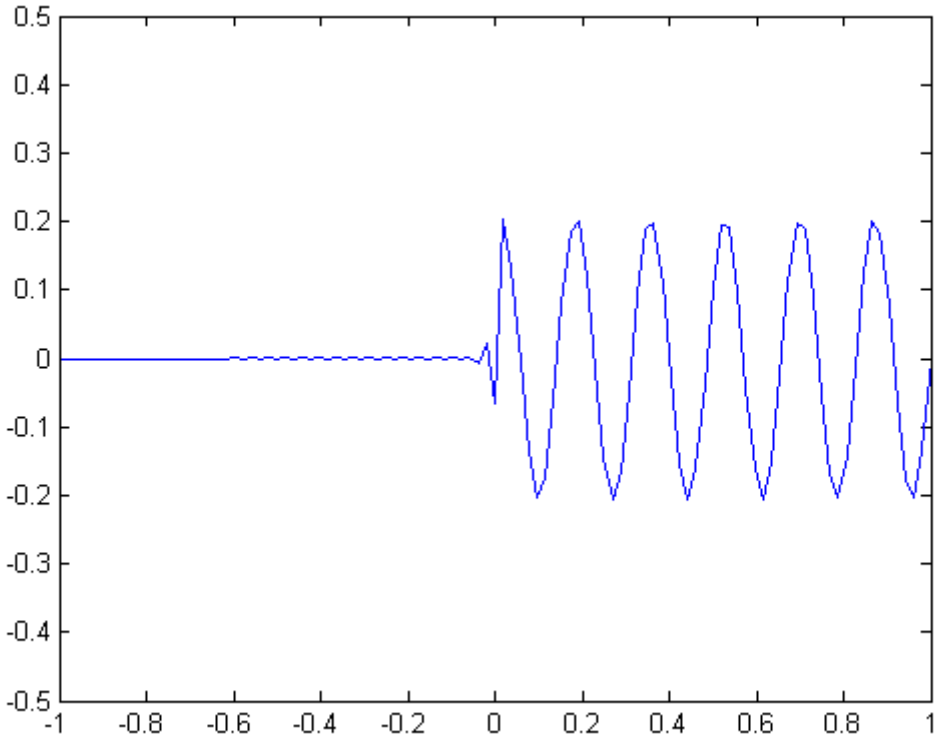

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

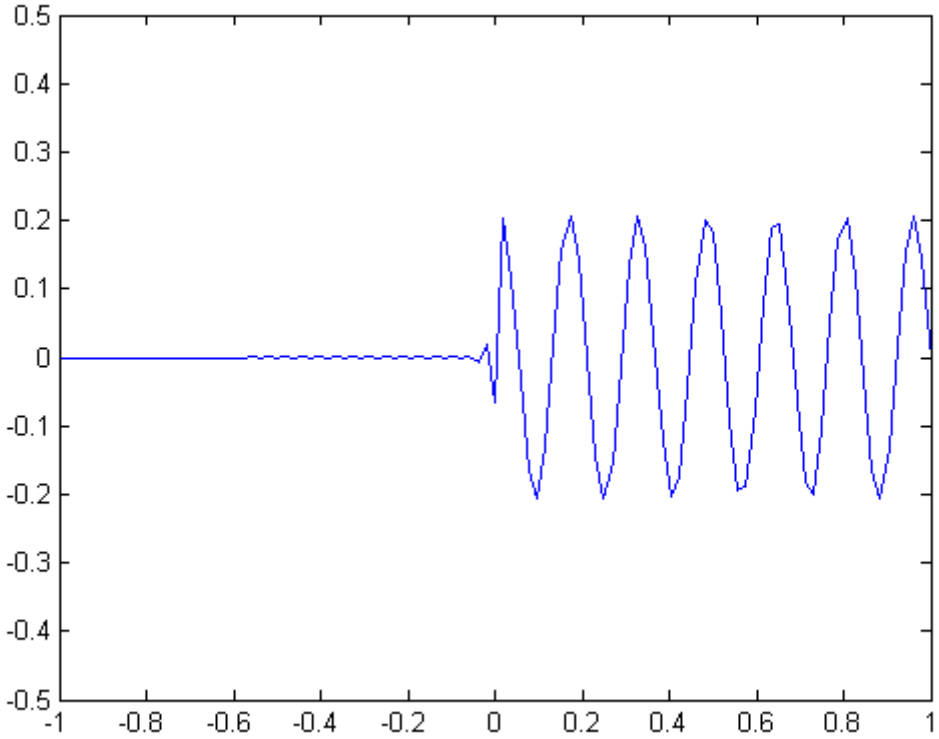

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

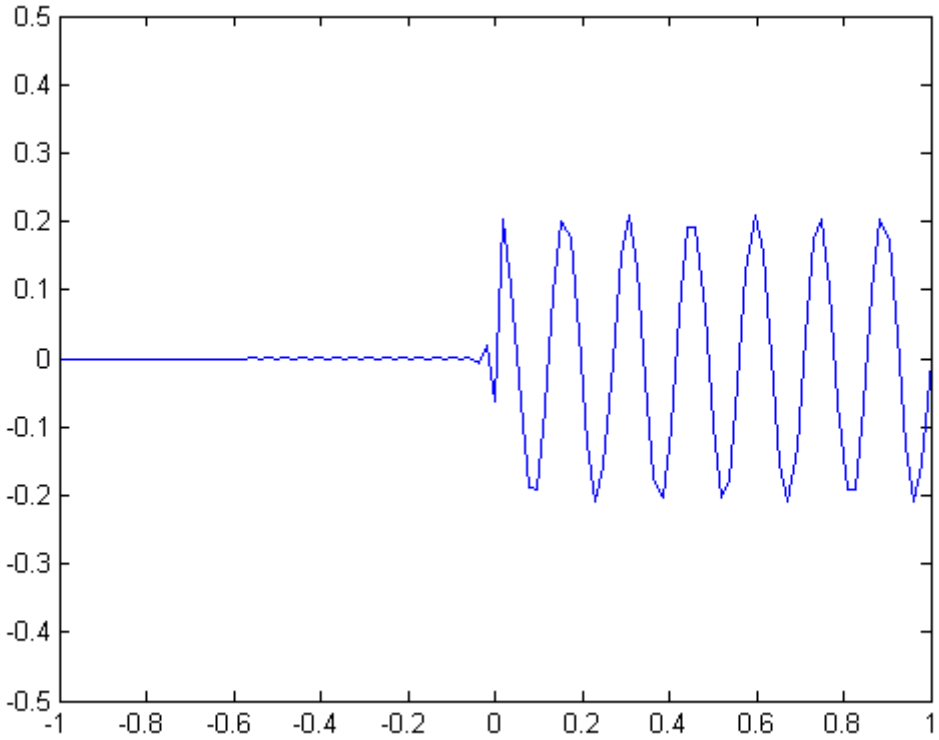

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

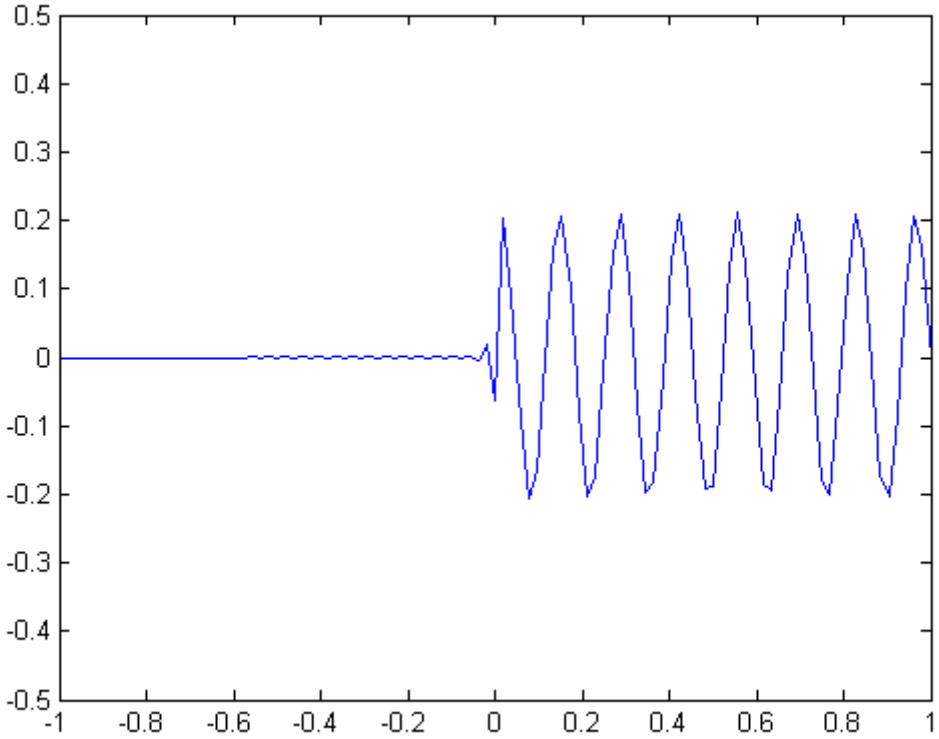

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

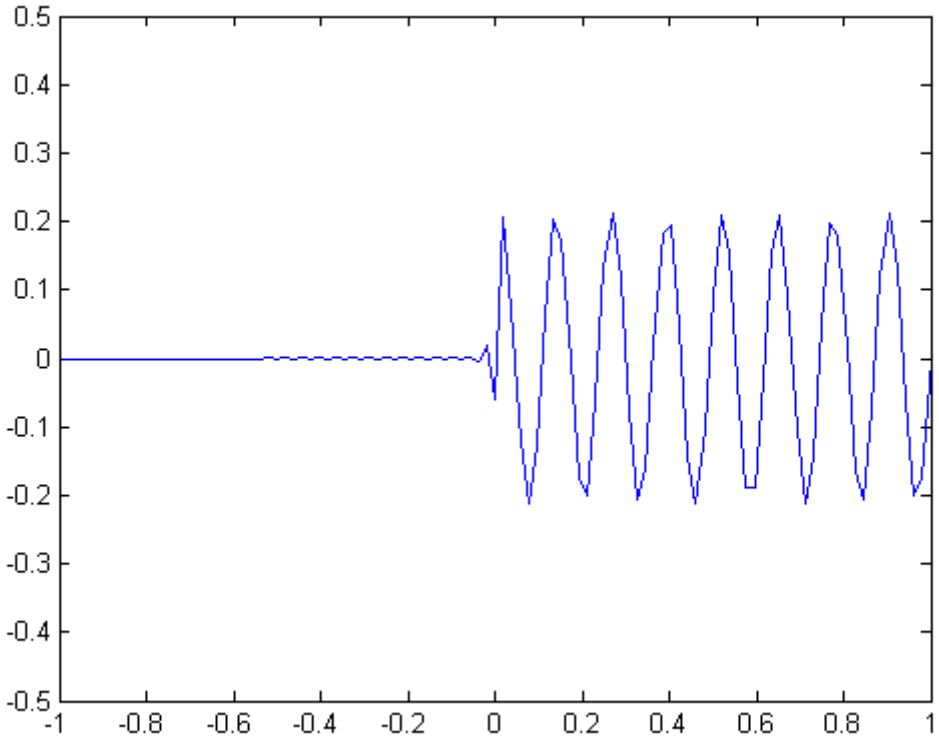

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

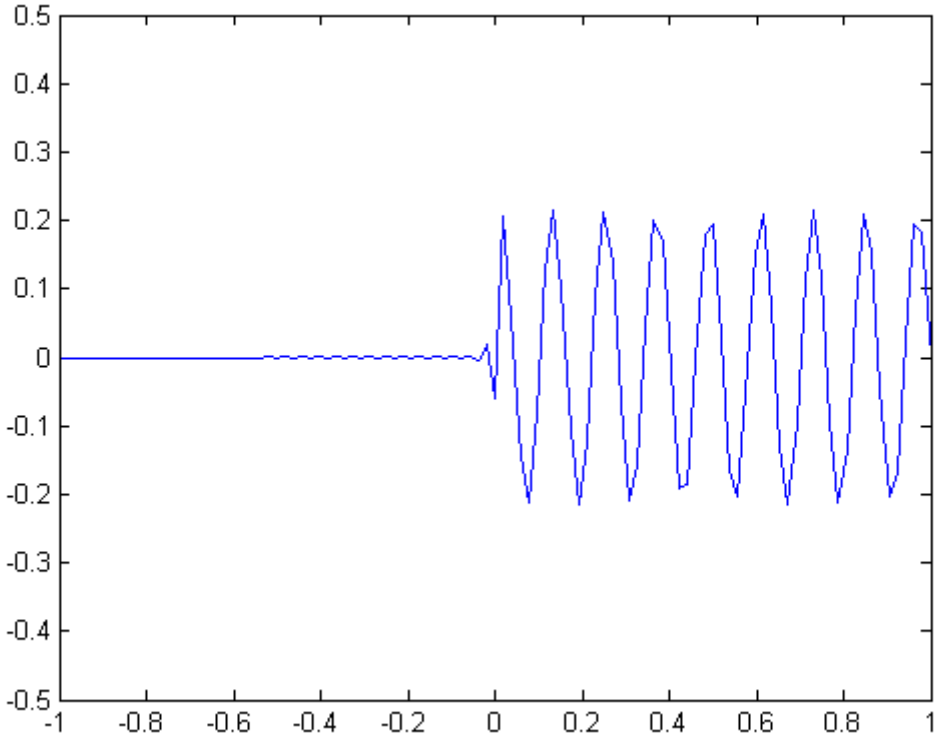

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

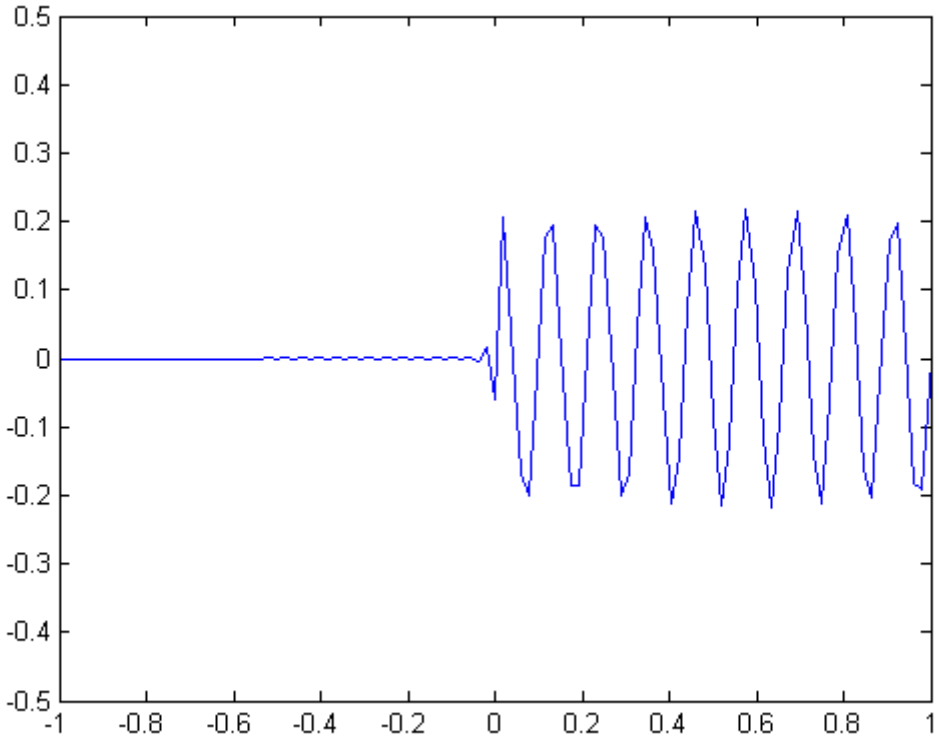

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

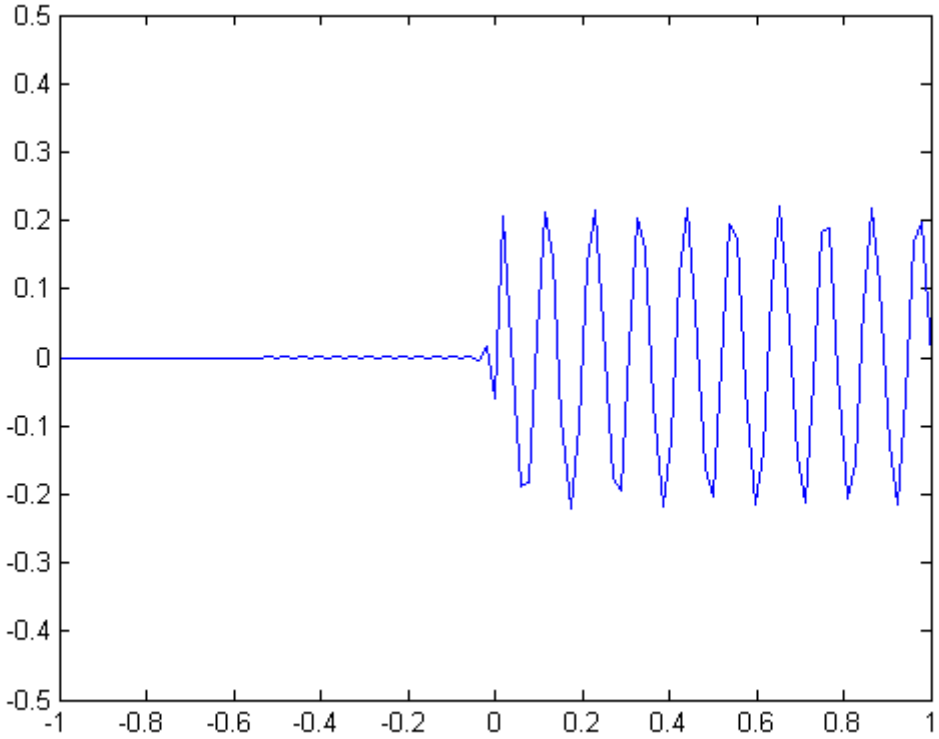

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

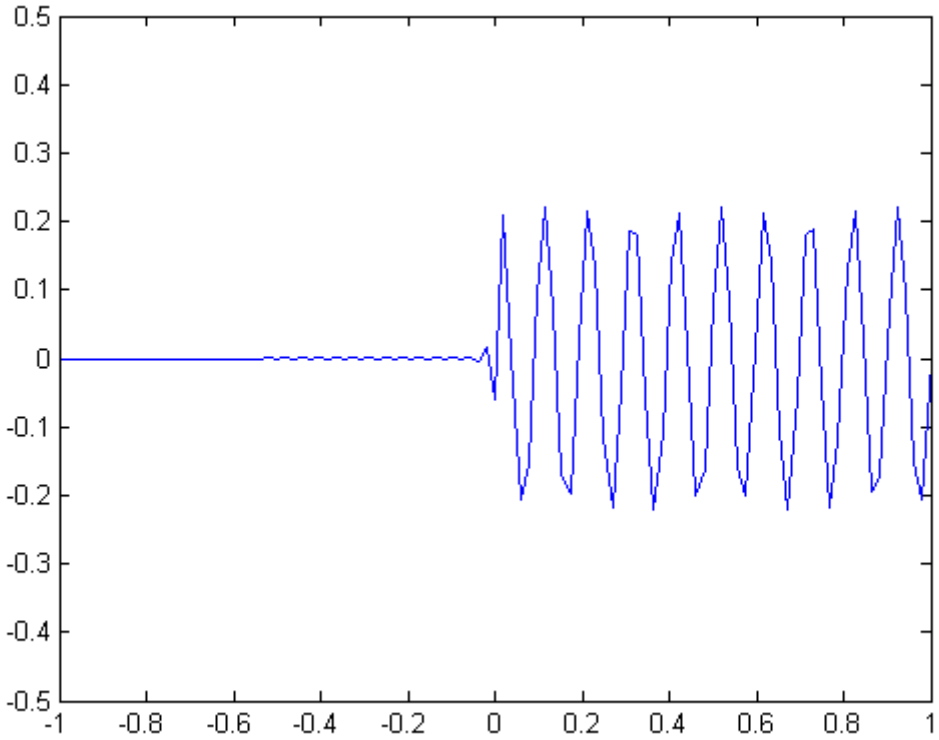

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

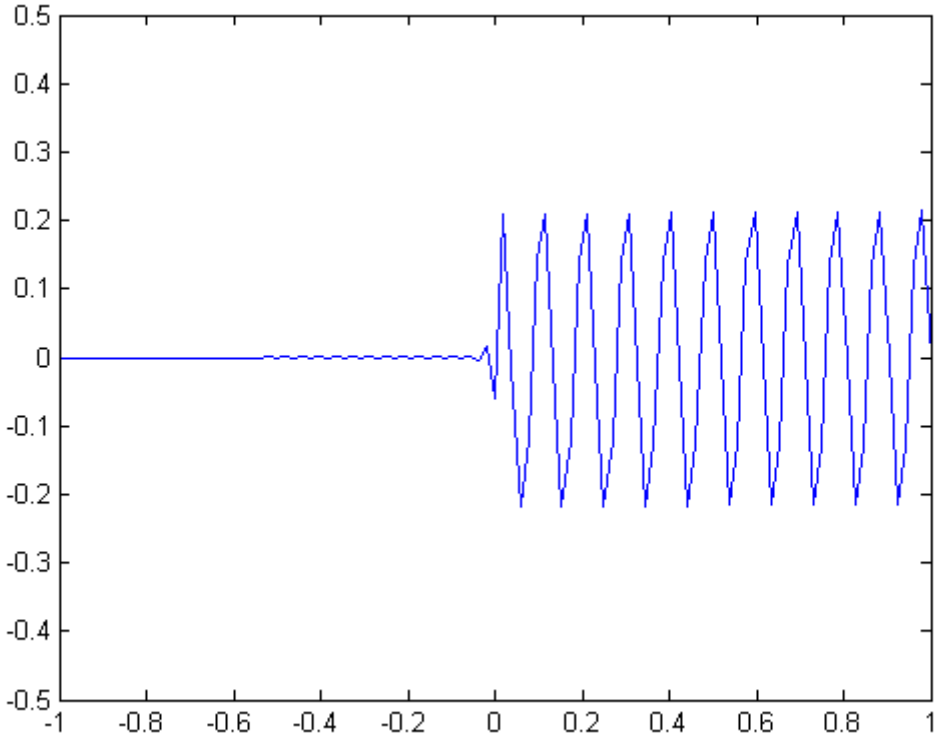

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

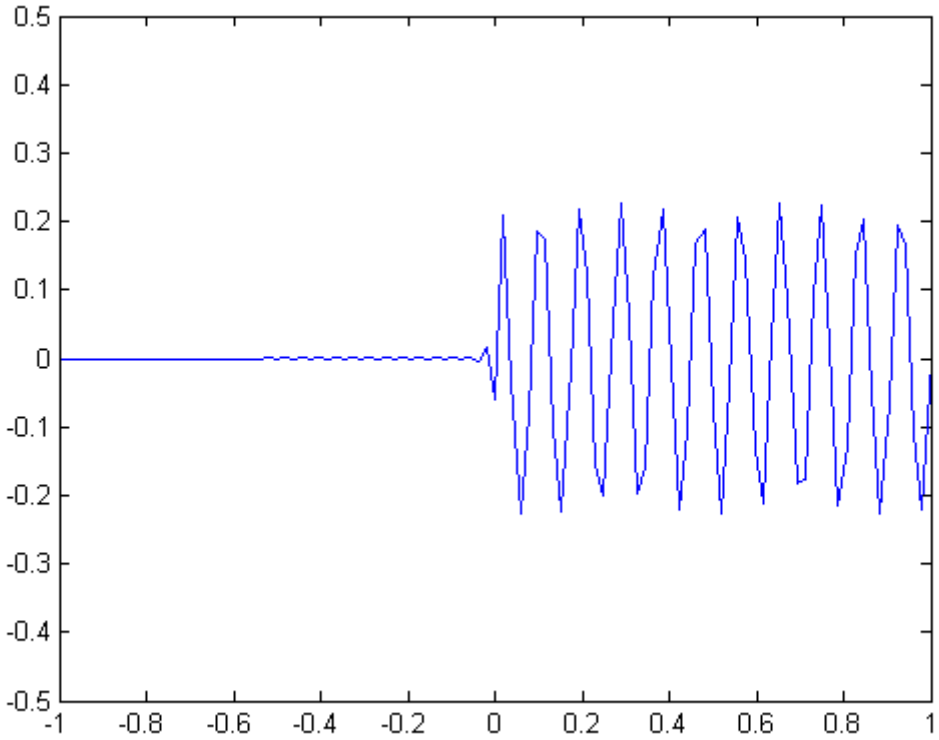

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

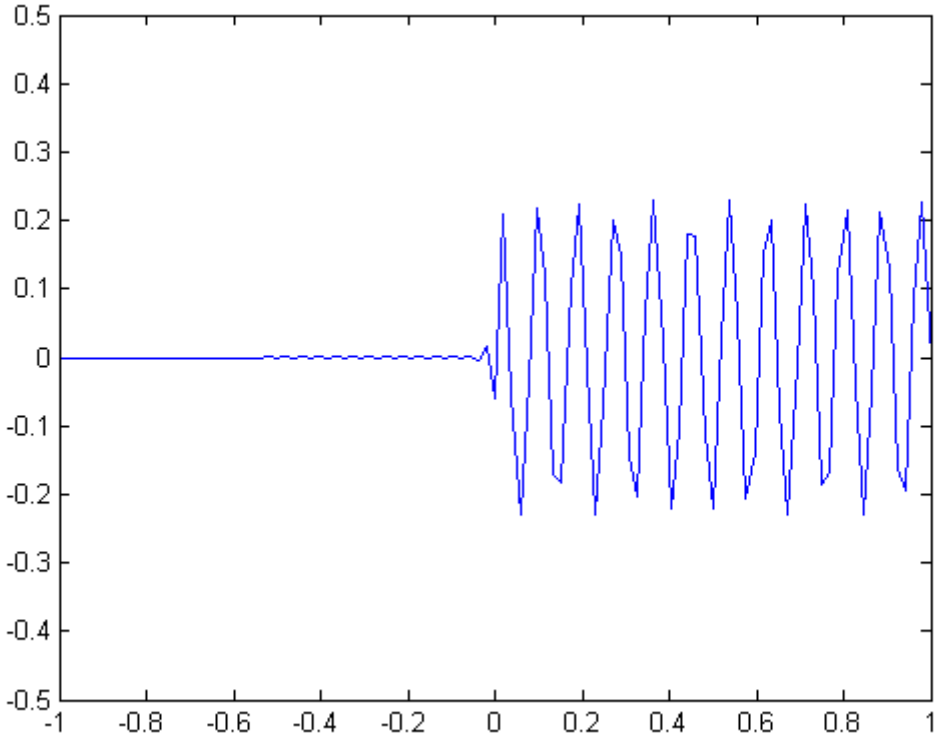

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

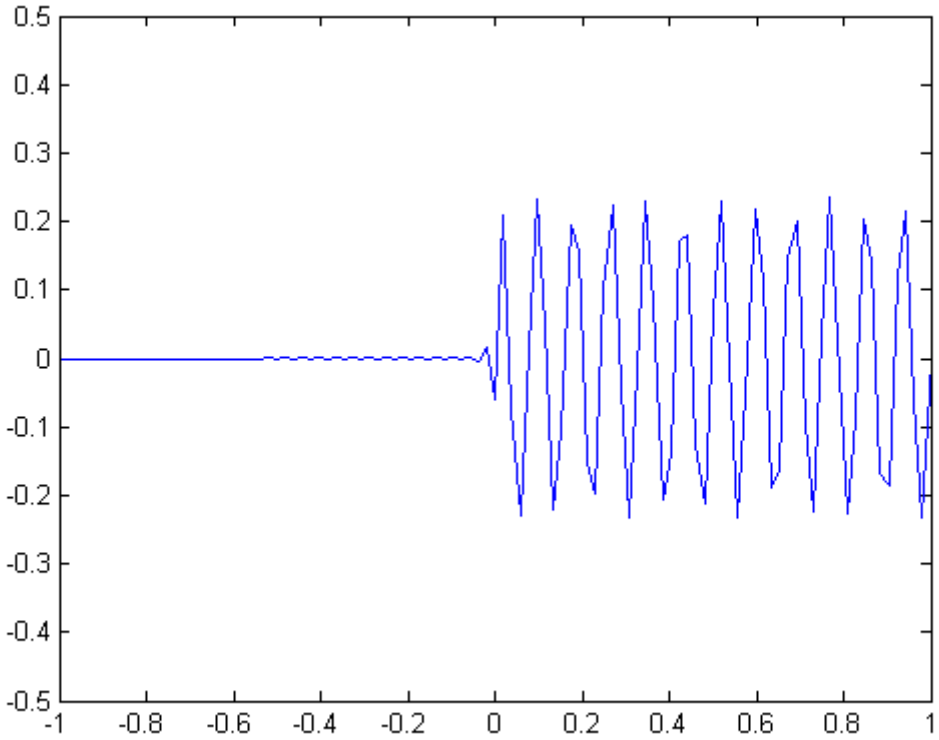

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

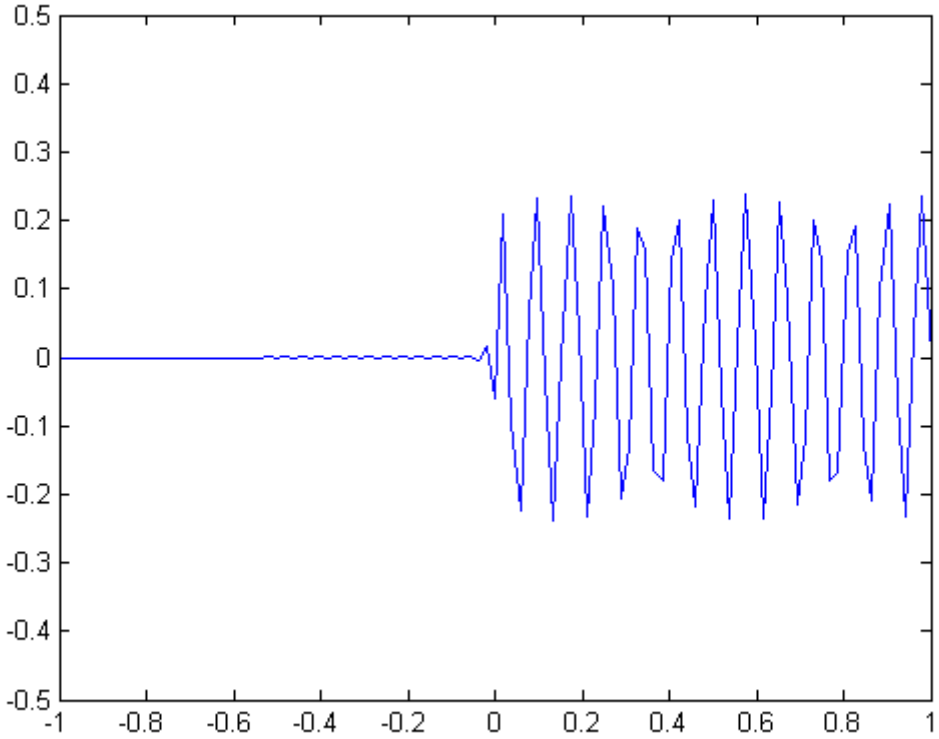

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

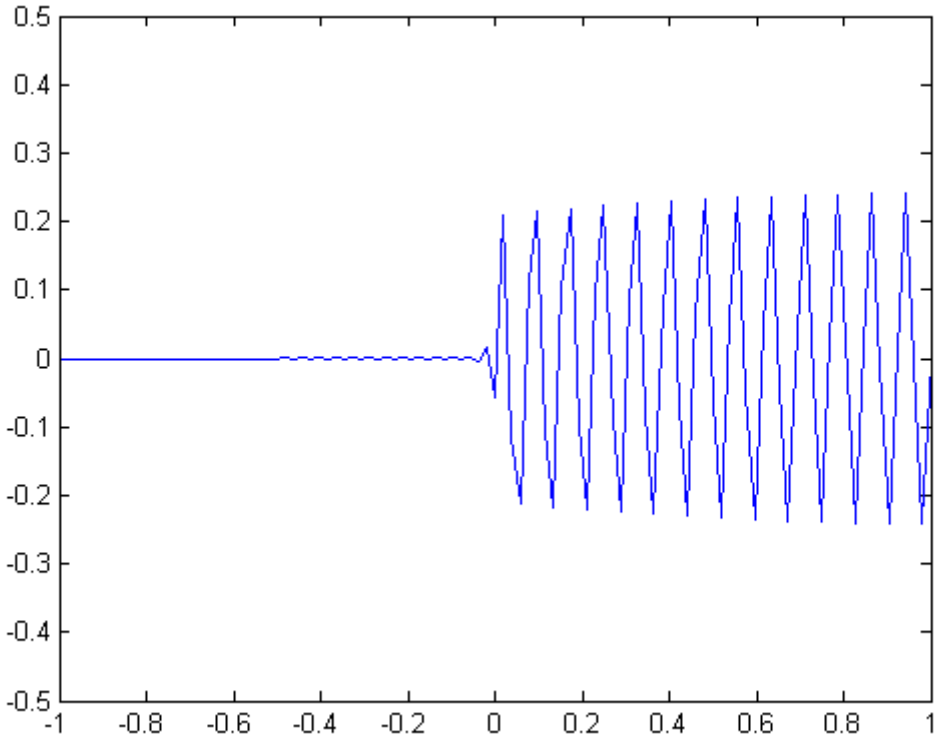

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

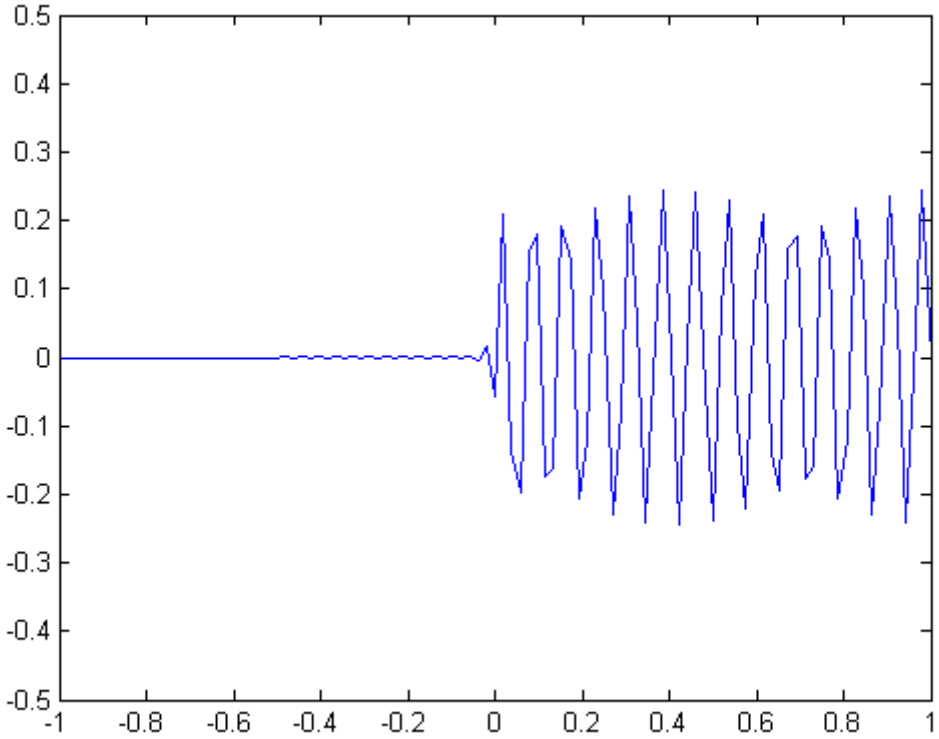

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

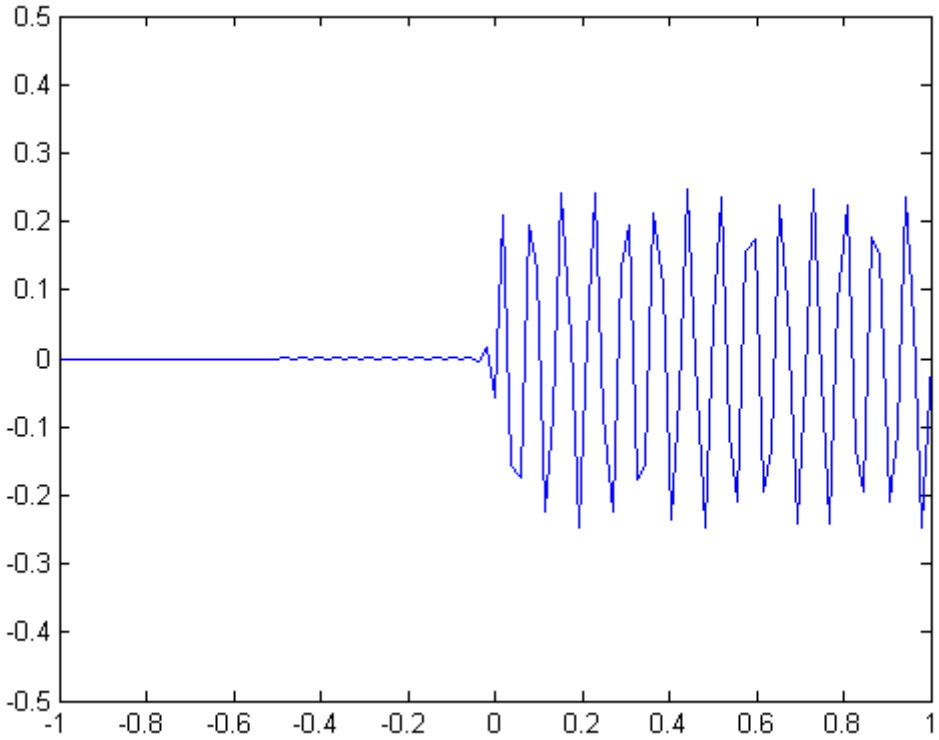

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

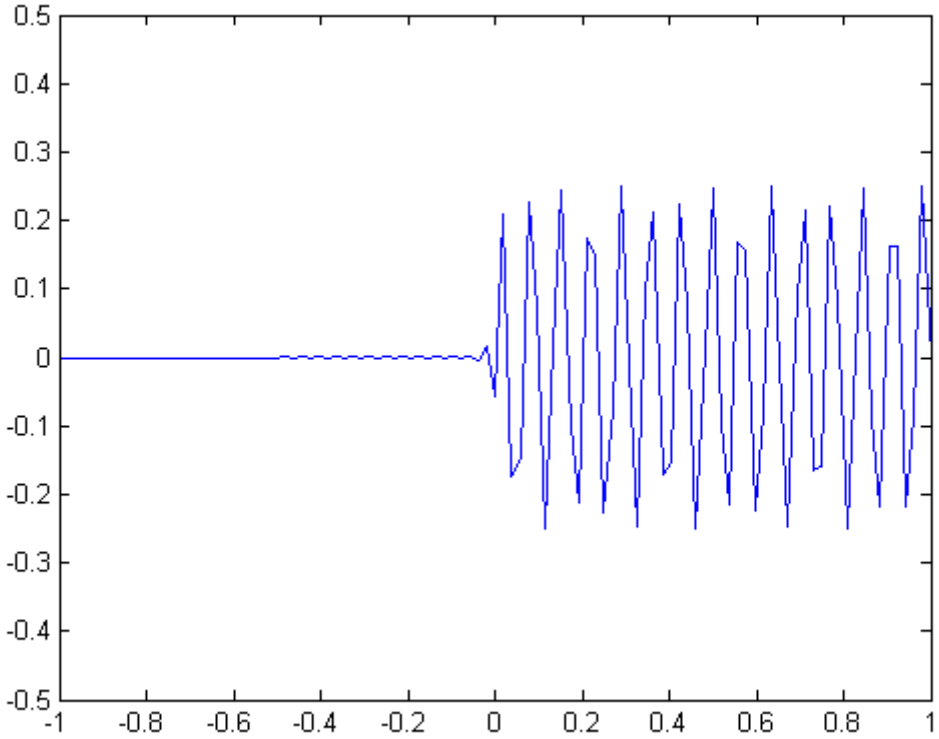

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

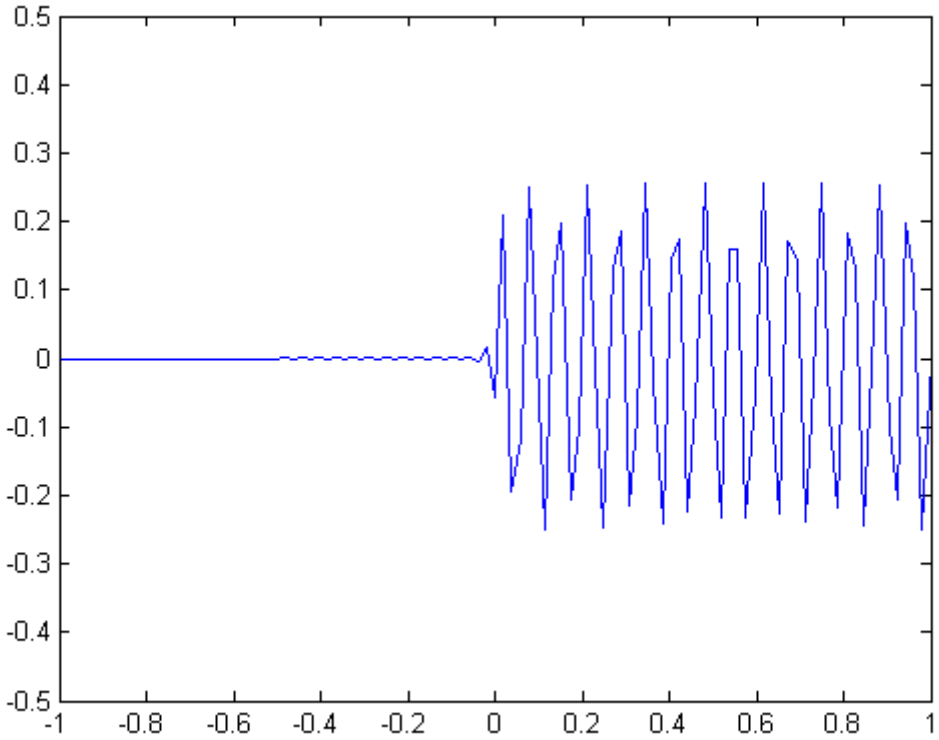

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

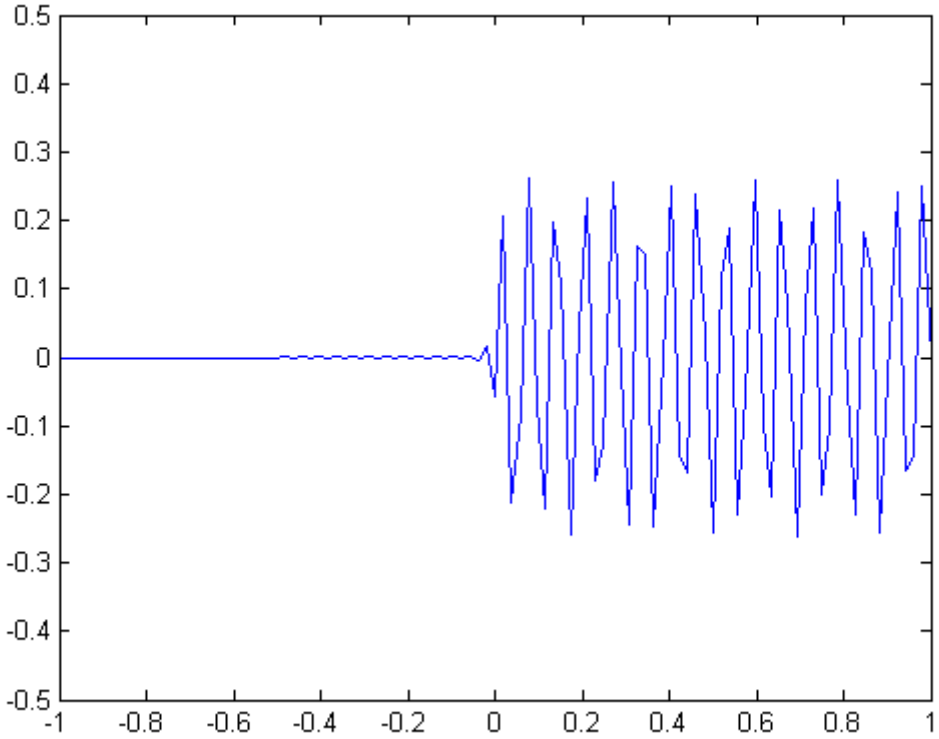

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

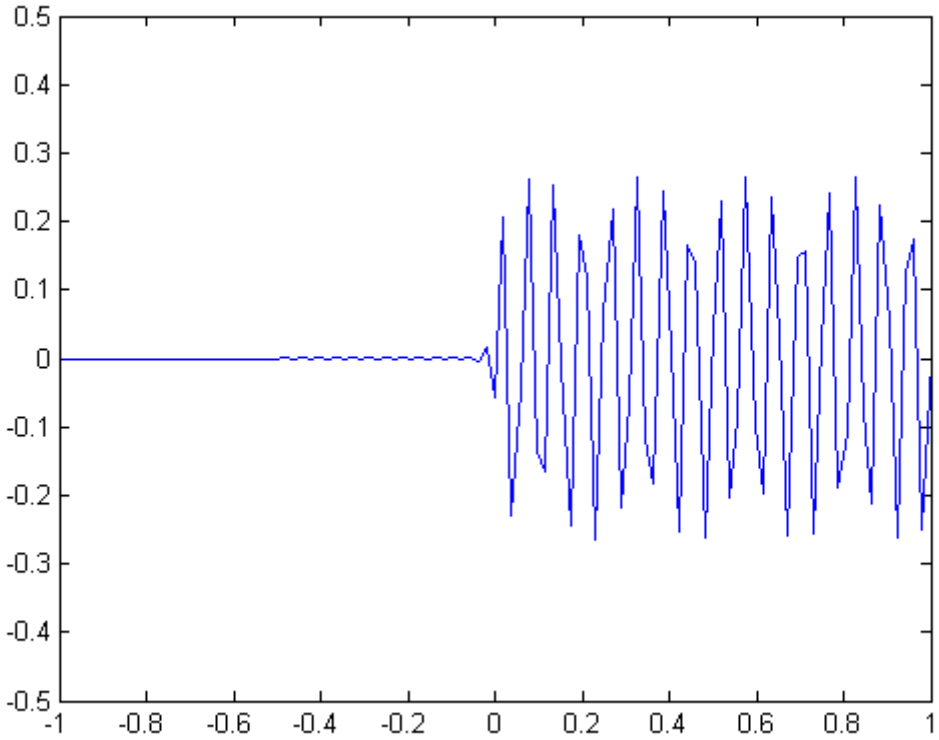

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

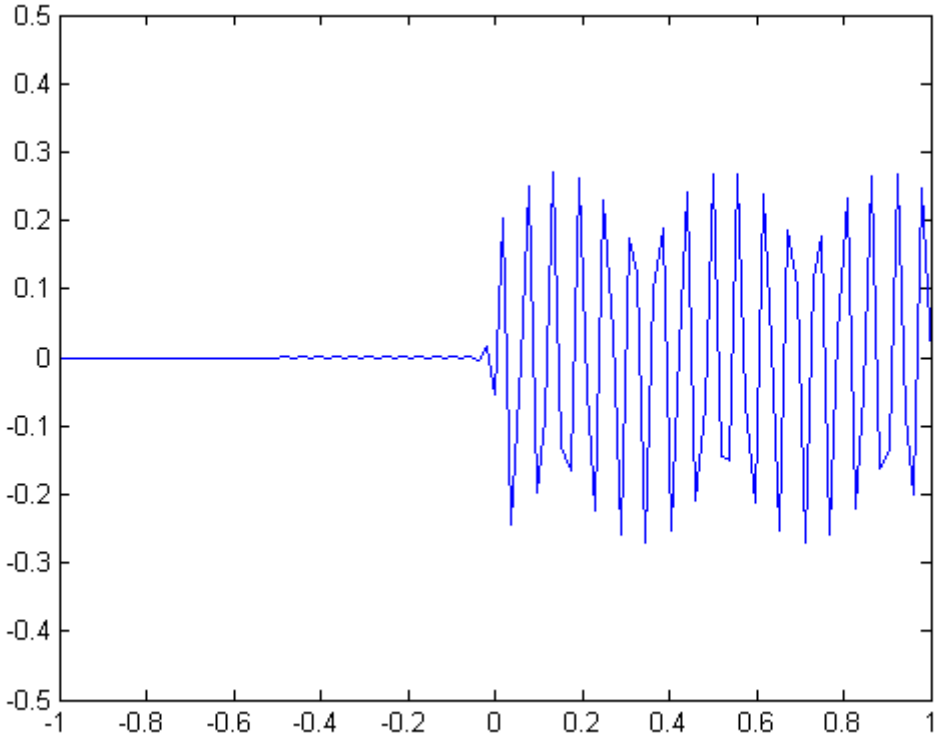

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

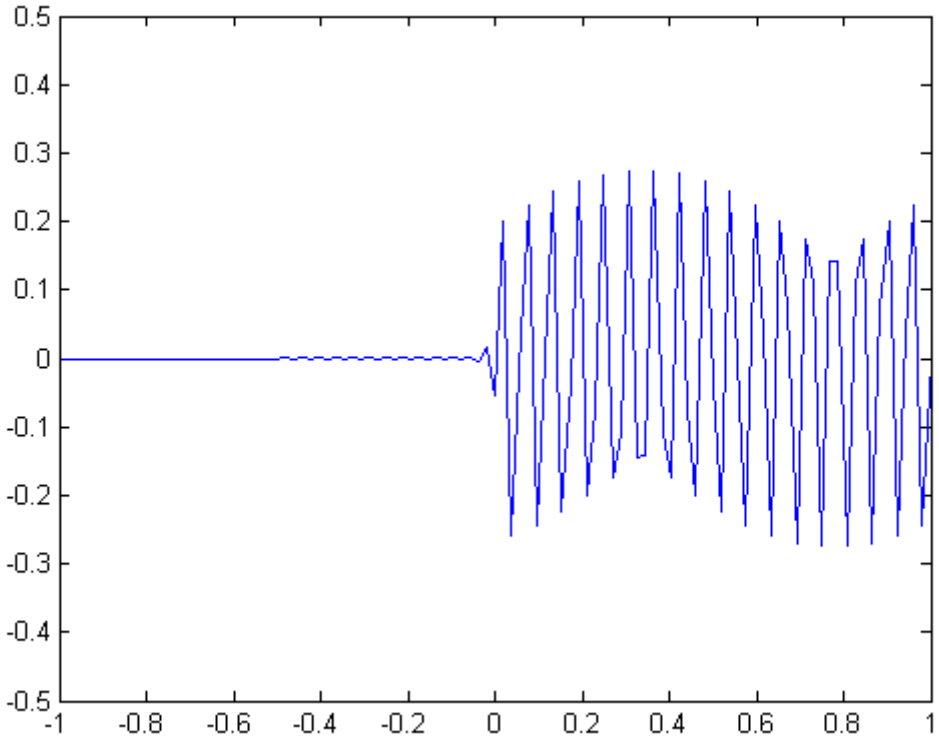

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

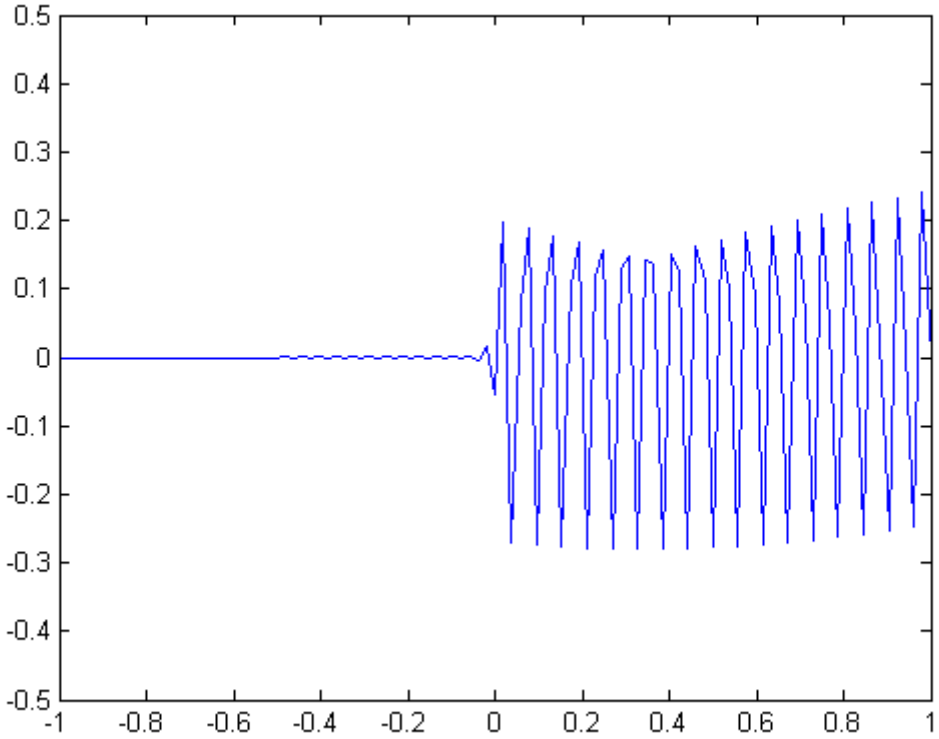

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

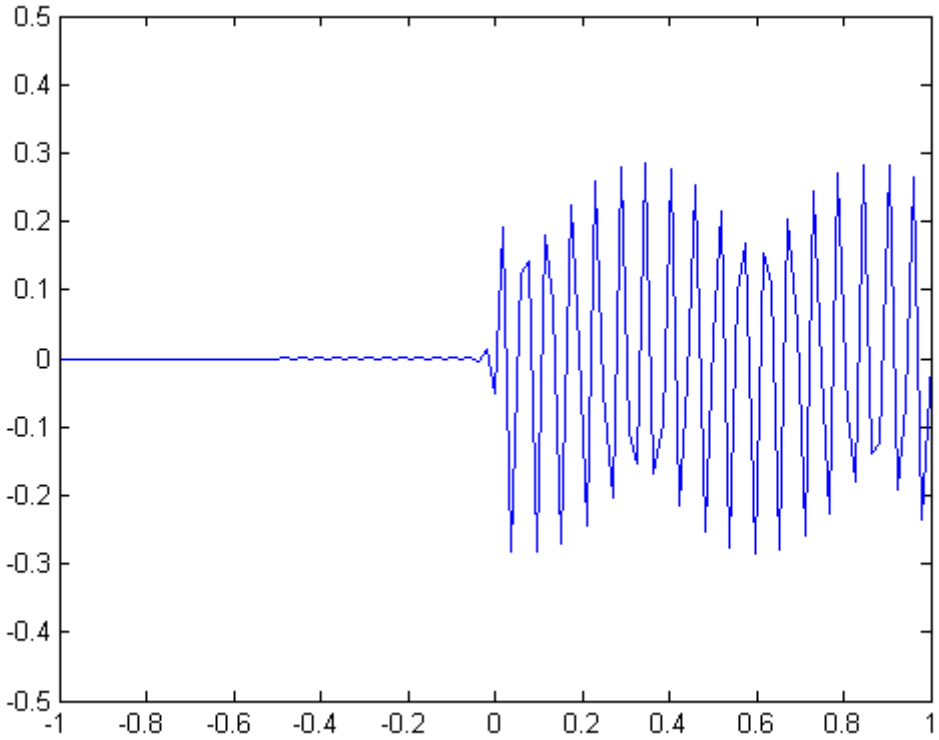

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

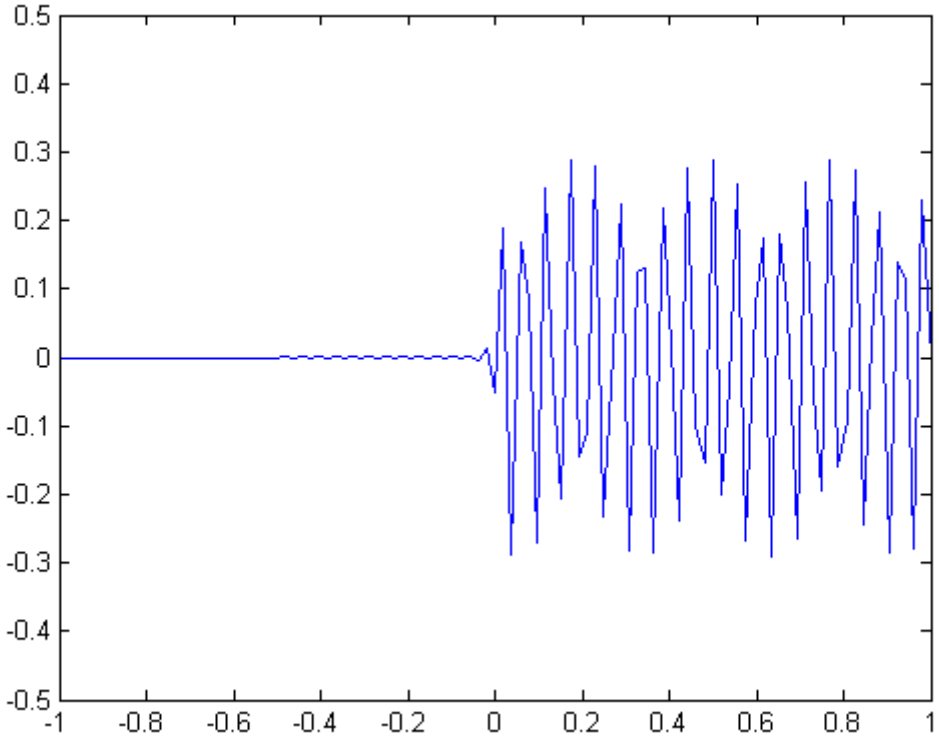

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

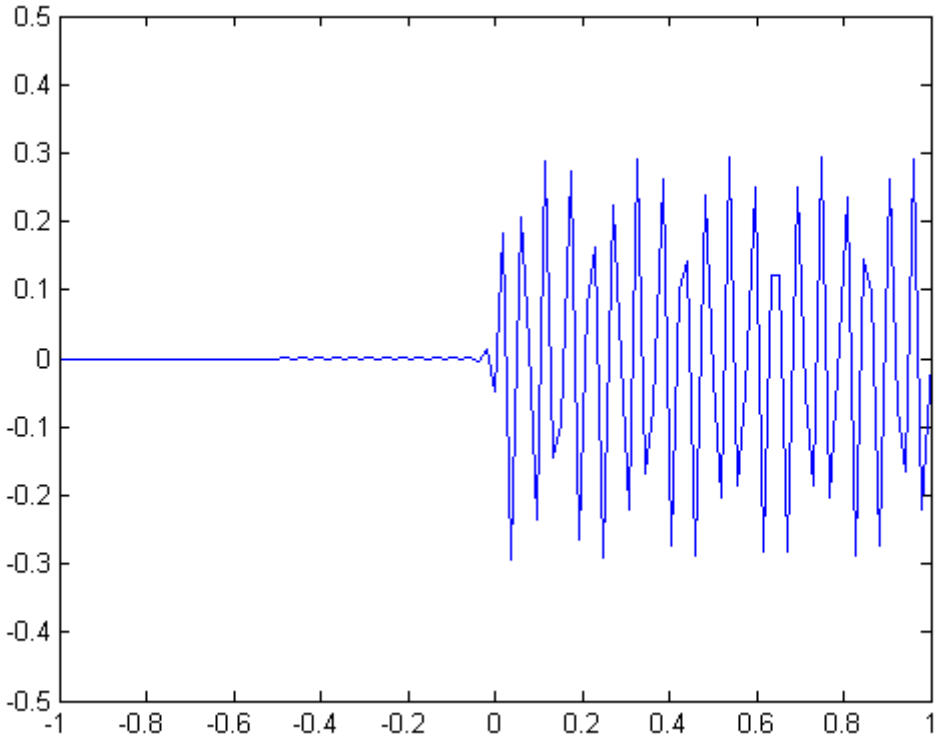

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

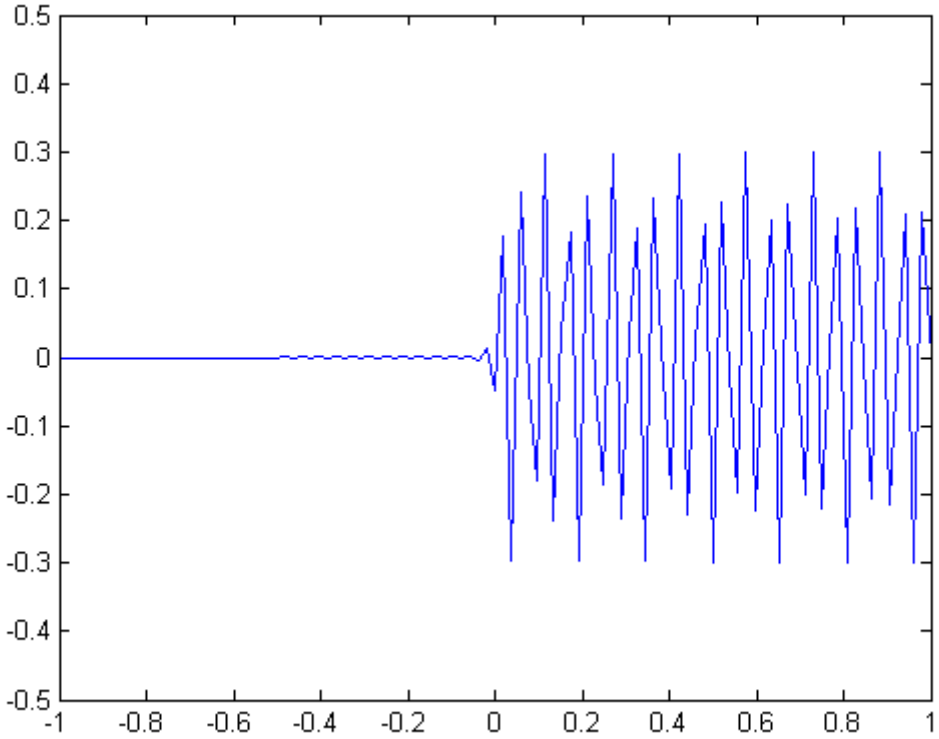

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

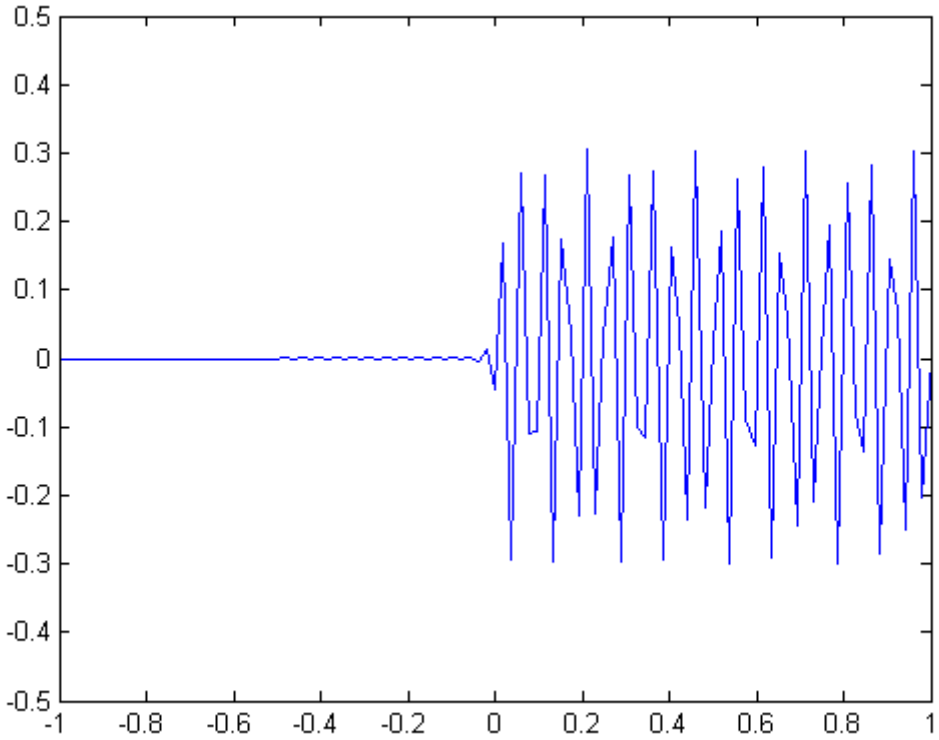

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

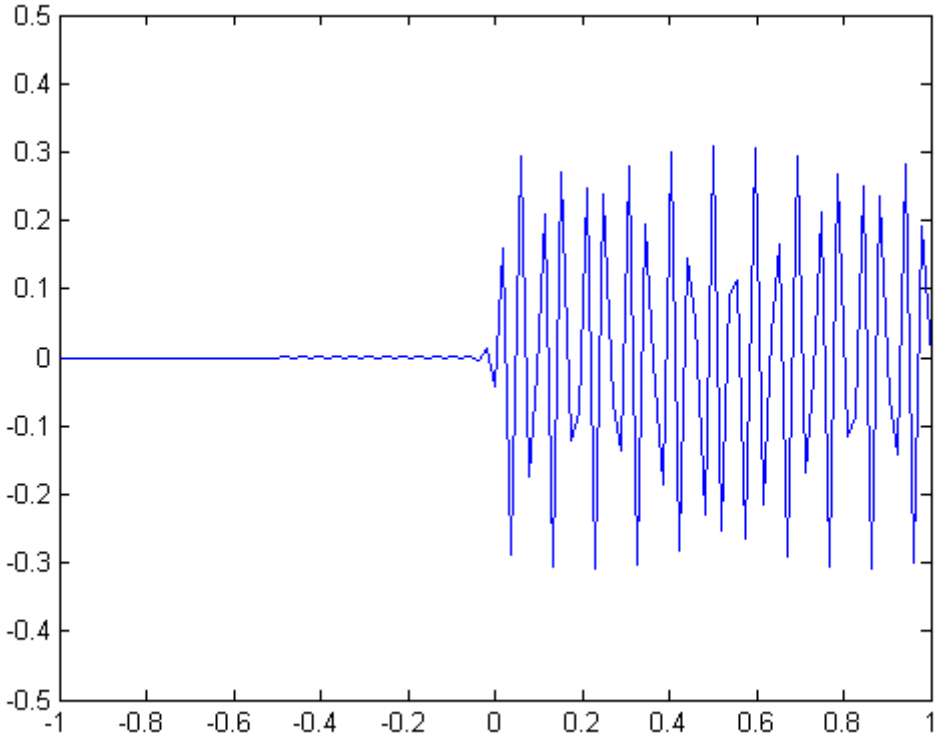

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

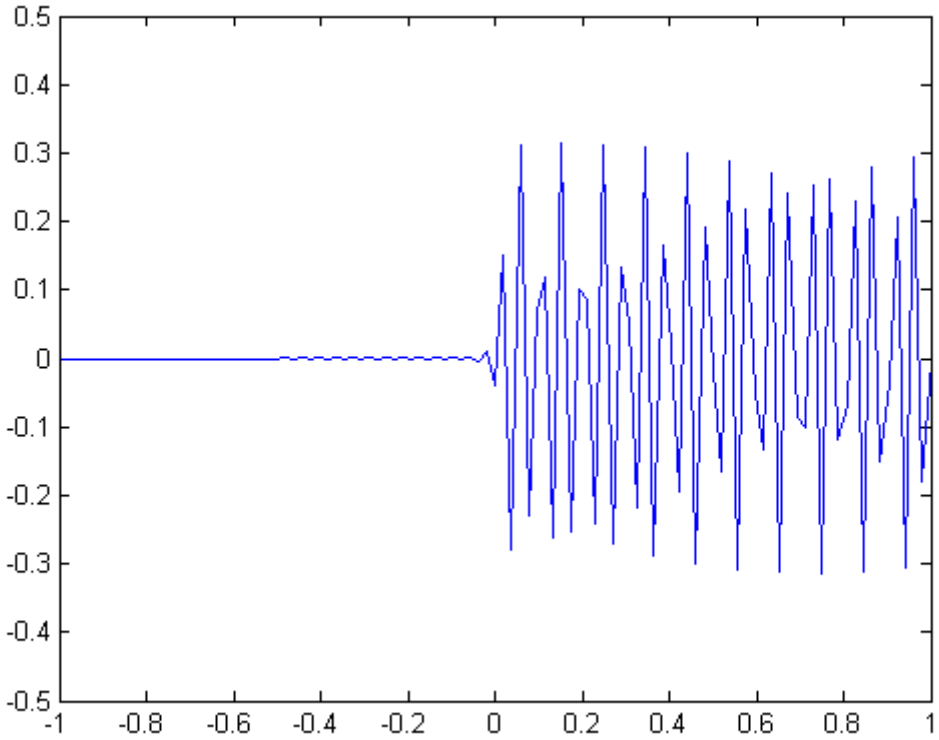

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

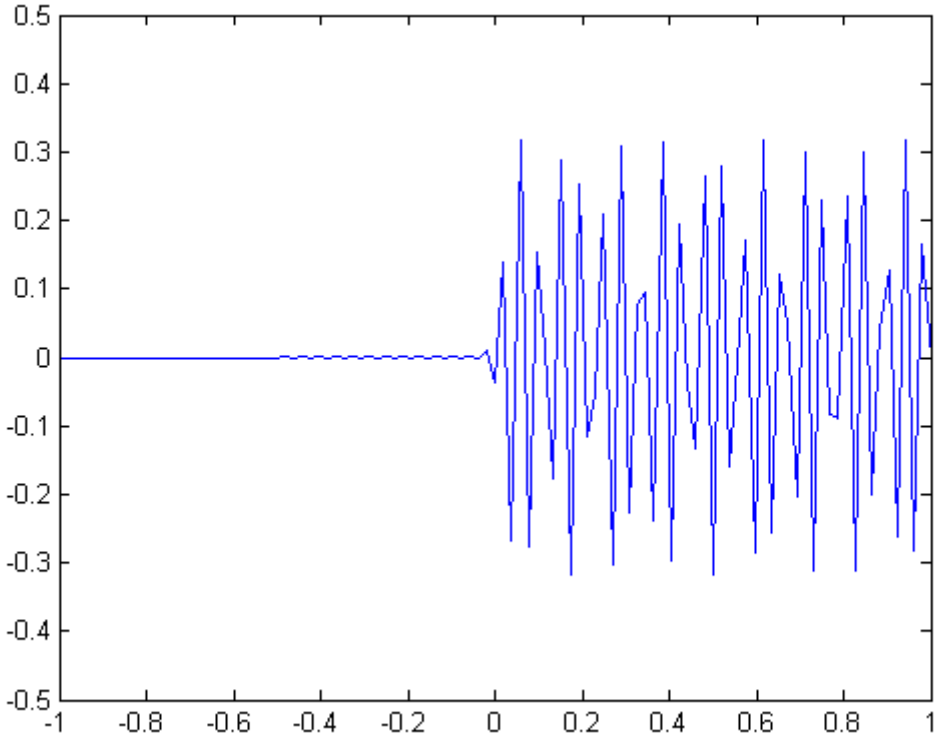

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

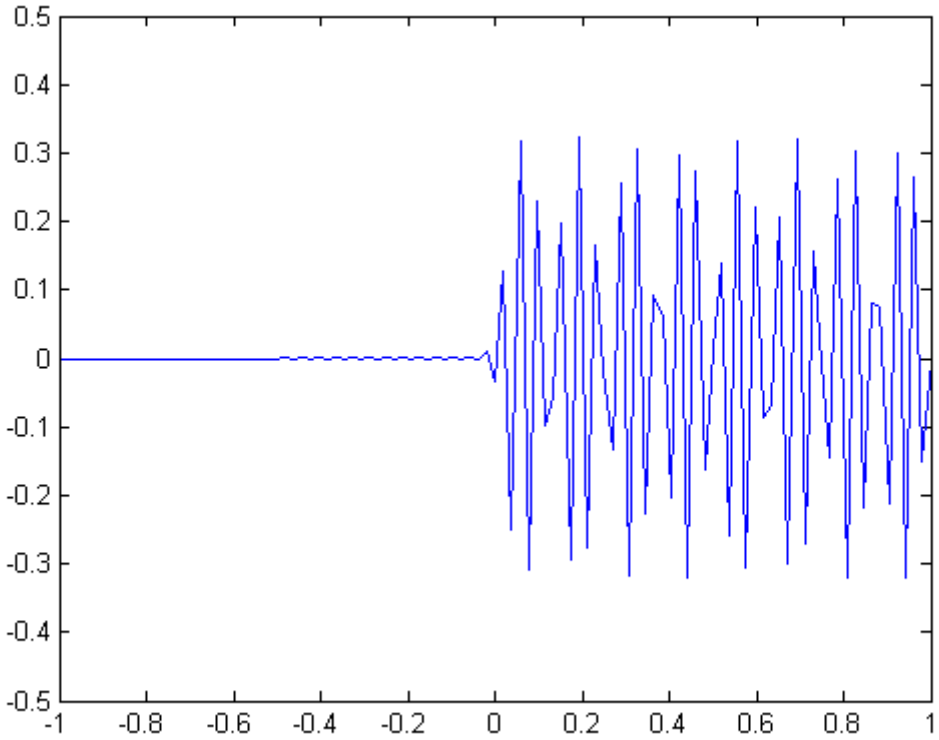

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

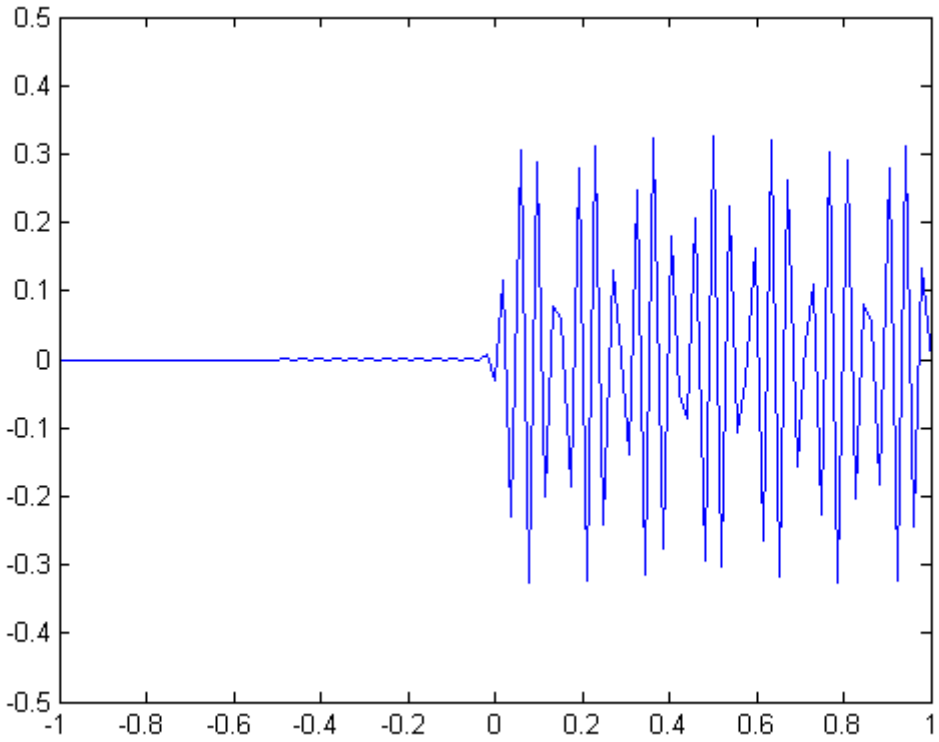

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

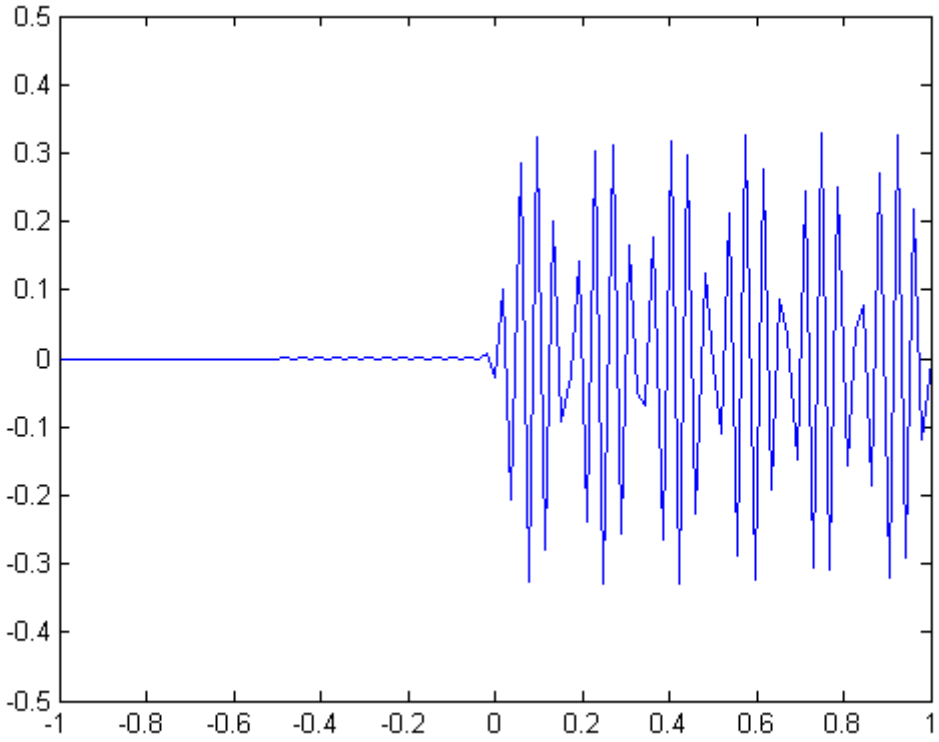

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

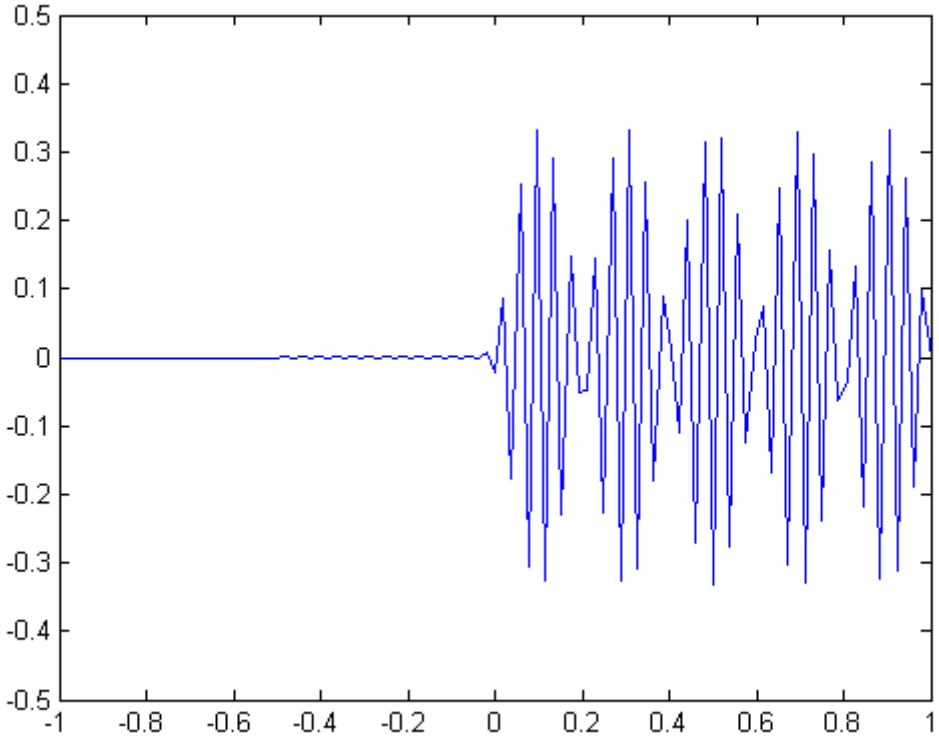

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

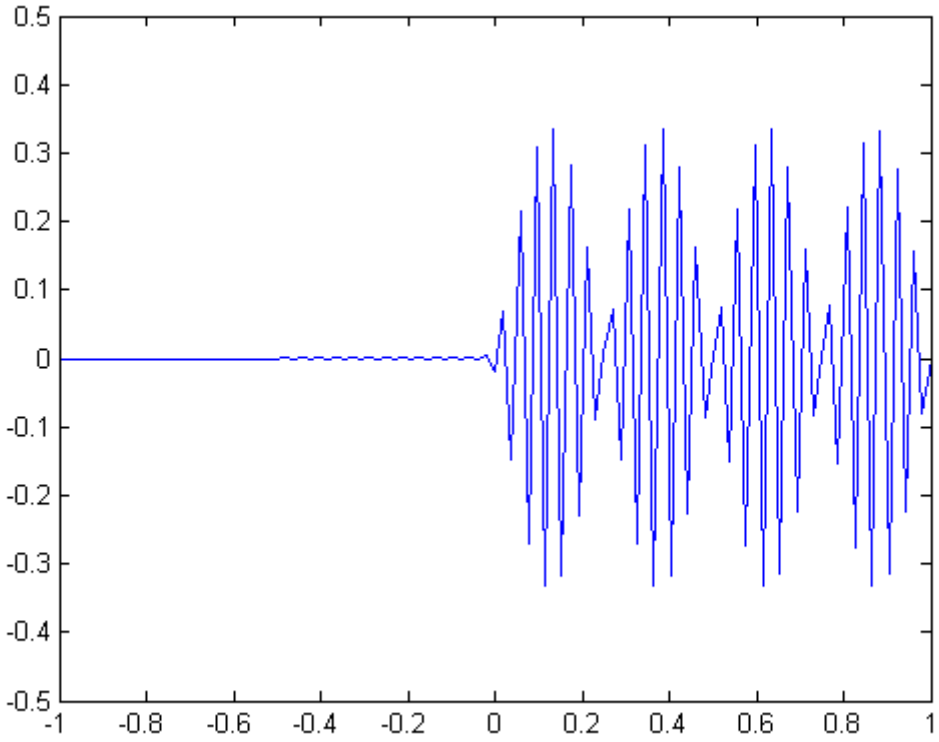

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

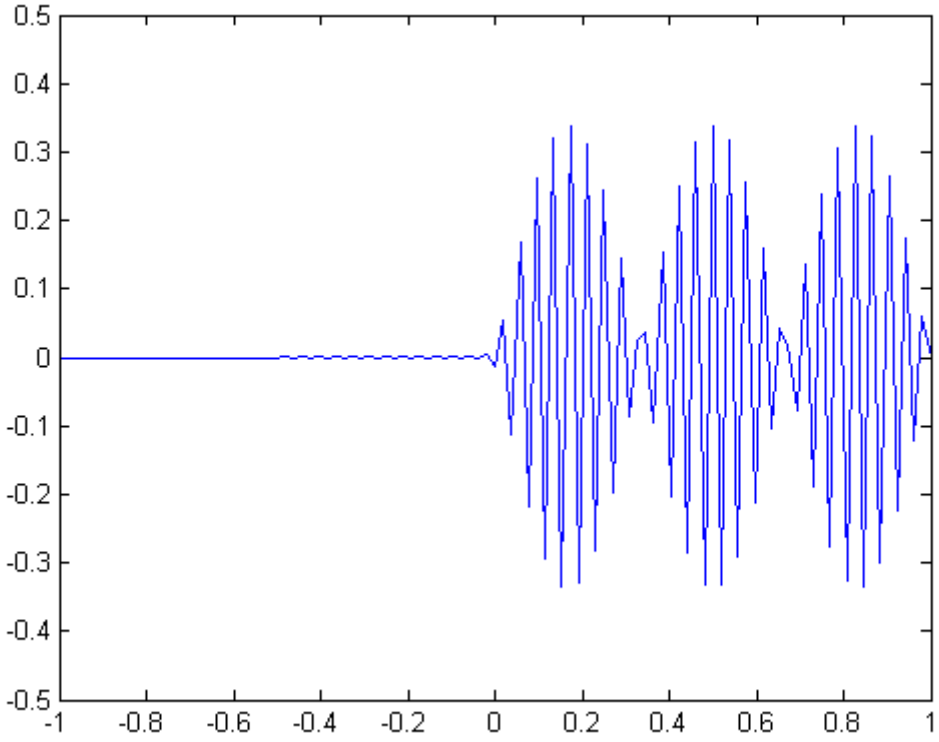

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

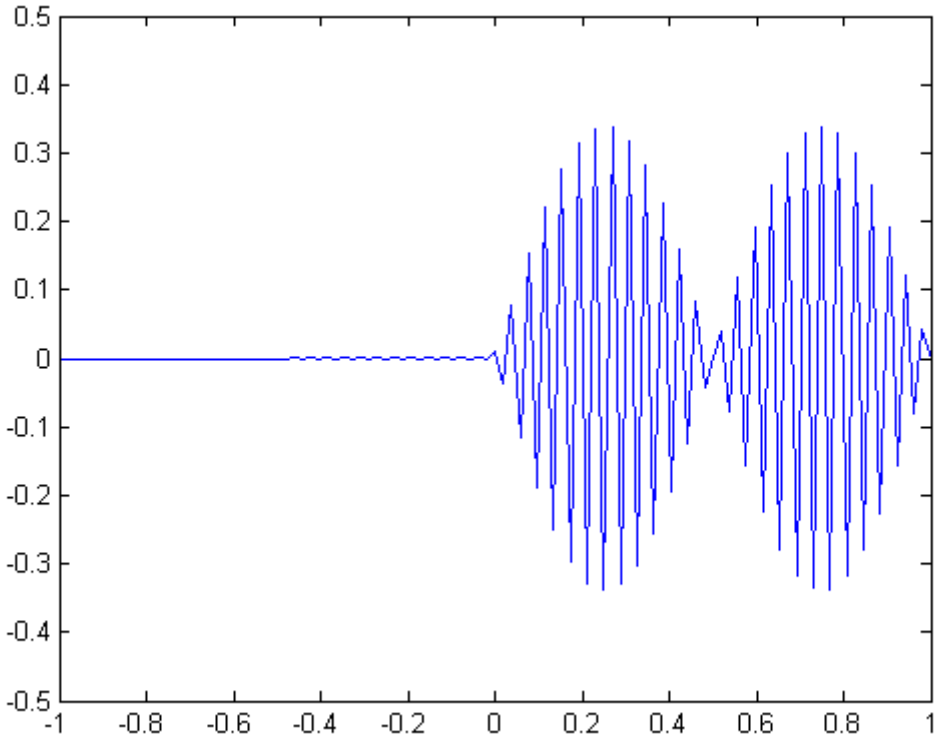

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

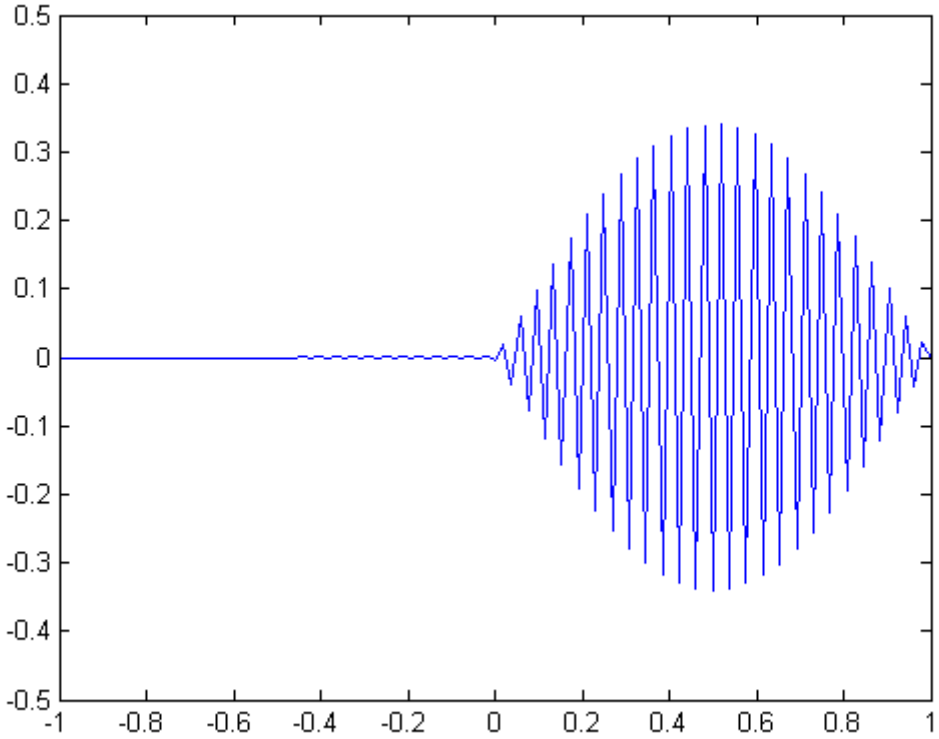

0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1



% disminuyendo h (aumenta n) y e,

% se tiene una discretización más fina en [-1,1] y los dos valores propios primeros y los dos últimos y sus respectivas gráficas se tienen con

>>fvpstiff(100,0.001) *% h=1/101, e=0.001*

valorpropio1 =

0.0097

valorpropio2 =

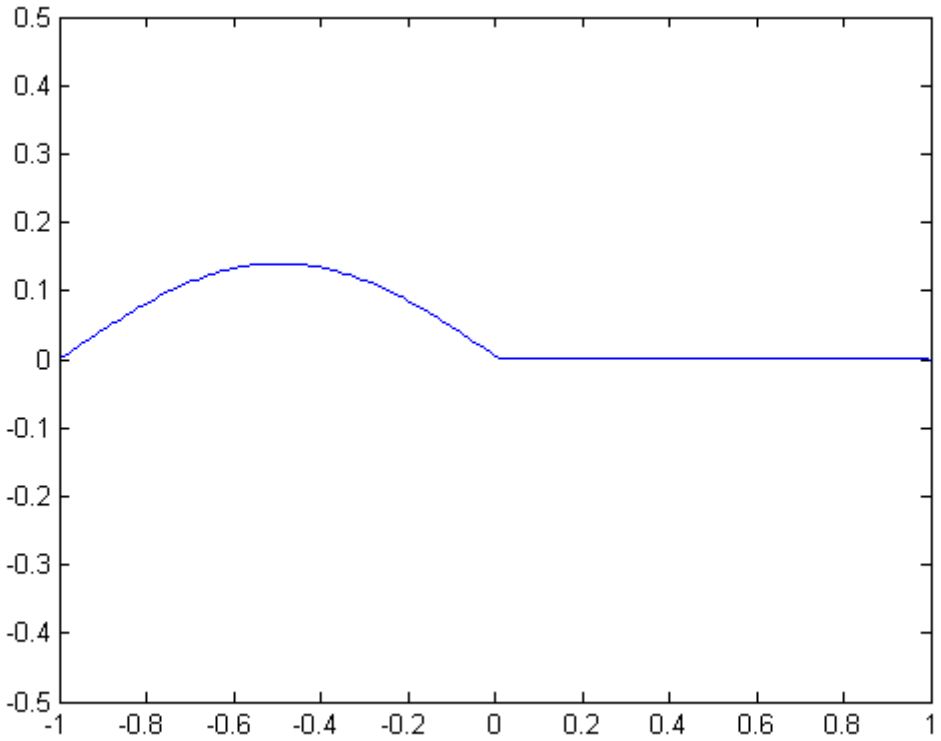
0.0386

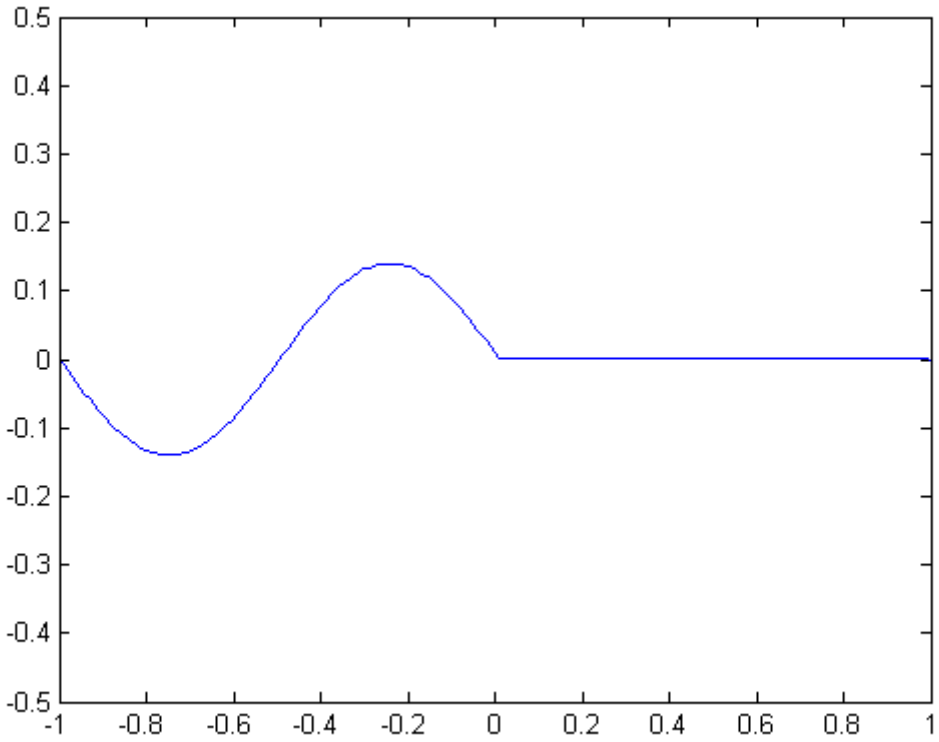
valorpropio200 =

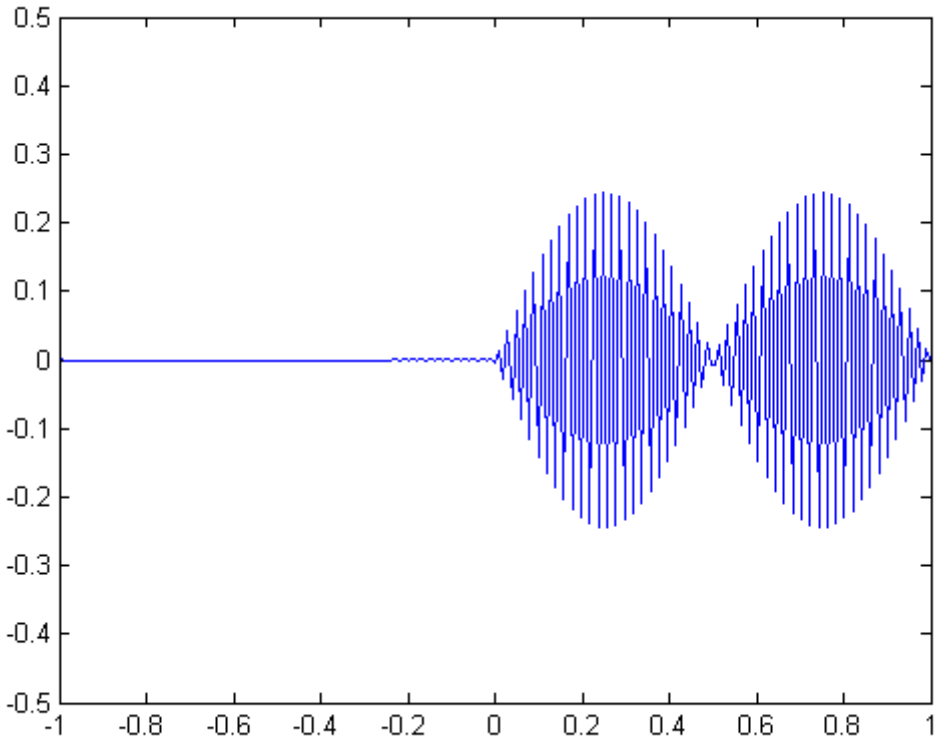
1.2206e+005

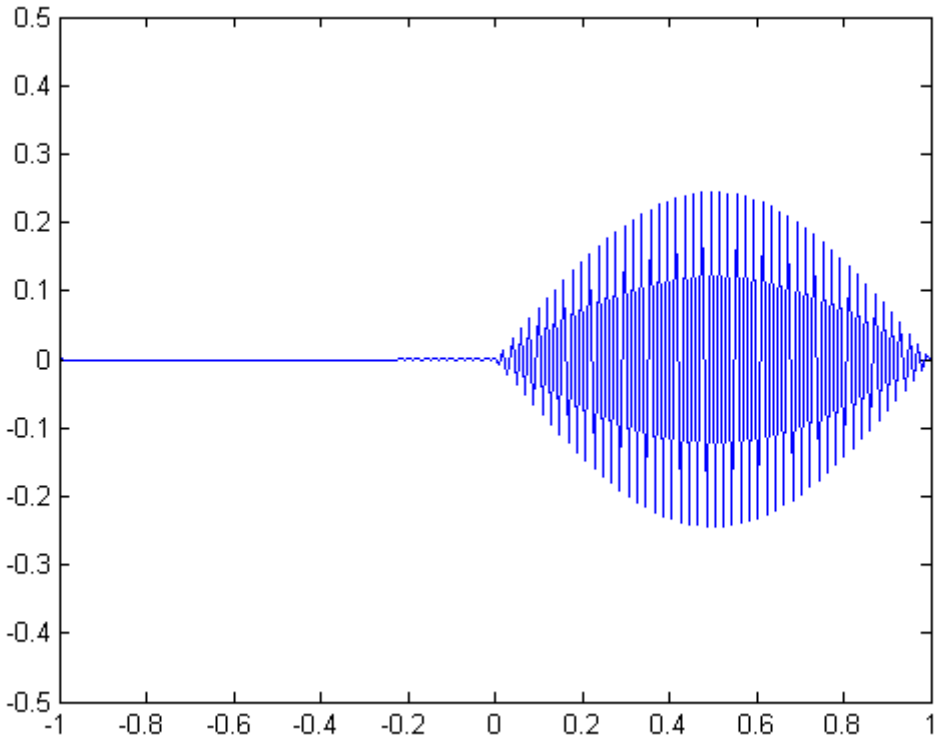
valorpropio201 =

1.2232e+005




0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1