Calling Fortran subroutines from kdb+

John Ludlow

February 24, 2011

1 Introduction

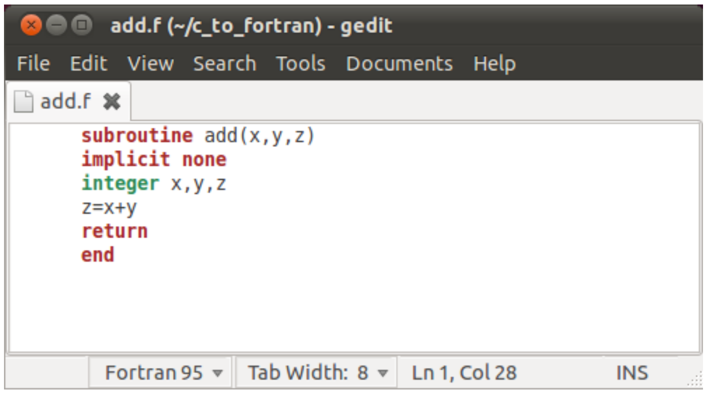
The scientific community has developed many useful, freely available numerical routines over the decades in Fortran. To make these routines accessible to the kdb+ user is the object of this report.

We will first use a simple example involving the addition of two integers in a Fortran subroutine to illustrate the process of calling a Fortran routine from kdb+. A more complex case involving the fourier transform of a list of floats will then be examined.

This work was carried out on a PC running Windows 7, with Ubuntu 10.10 installed under virtualbox, with the C and Fortran codes compiled using gcc version 4.4.5. and the fort77 package. The fort77 package provides an interface to the f2c program that converts Fortran code to C code. Other Fortran compilers such as gfortran are also available.

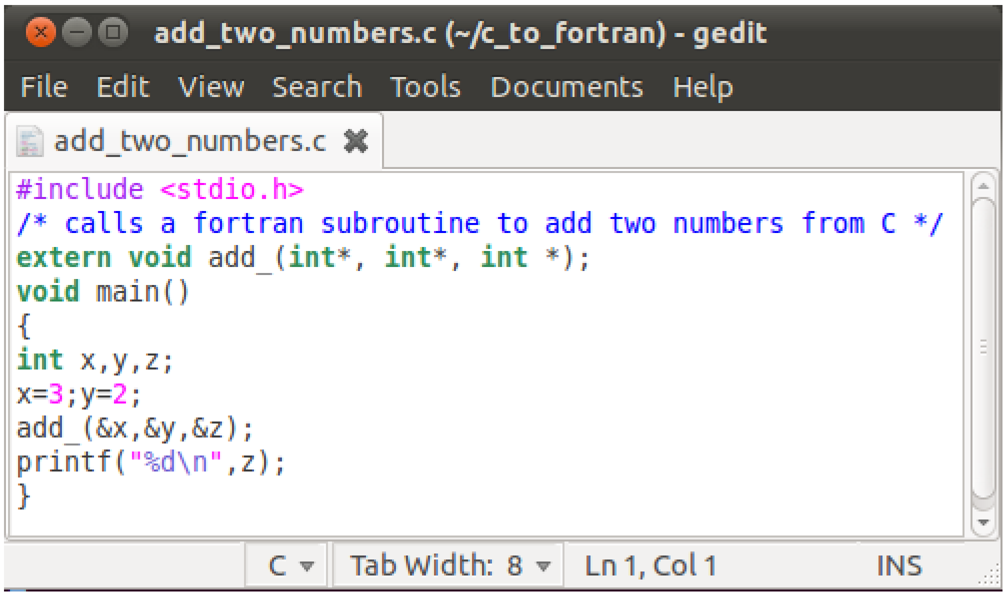
2 Calling a Fortran subroutine from C

- First we show how to use a C wrapper to call the Fortran subroutine add.f. This routine simply adds two numbers that are supplied to it:


add.f (~/c_to_fortran) - gedit
File Edit View Search Tools Documents Help
add.f
Fortran 95
Tab Width: 8
Ln 1, Col 28
INS

```
      subroutine add(x,y,z)
      implicit none
      integer x,y,z
      z=x+y
      return
      end
```

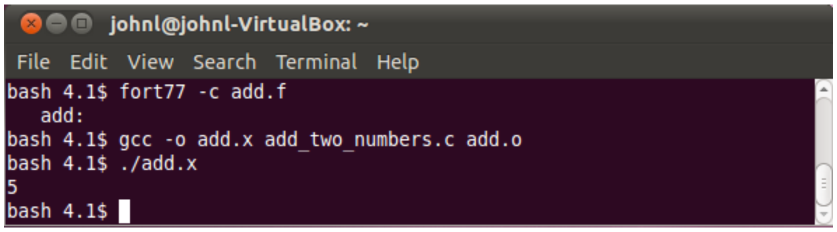
- Now we write a C wrapper to this subroutine, add_two_numbers.c :


add_two_numbers.c (~/c_to_fortran) - gedit
File Edit View Search Tools Documents Help
add_two_numbers.c
C
Tab Width: 8
Ln 1, Col 1
INS

```
#include <stdio.h>
/* calls a fortran subroutine to add two numbers from C */
extern void add_(int*, int*, int *);
void main()
{
int x,y,z;
x=3;y=2;
add_(&x,&y,&z);
printf("%d\n",z);
}
```

There are a few important points to note. First, arguments to Fortran subroutines are passed by reference rather than by value as in C. Therefore the Fortran subroutine is called in the C code with the memory addresses of the variables as arguments. Secondly, note the underscore after the name of the Fortran subroutine, This enables the gcc compiler to recognize the Fortran subroutine.

- Now we first compile the Fortran subroutine add.f using the fort77 compiler to produce an object file add.o. The C code add_two_numbers.c is then compiled with the object file add.o:

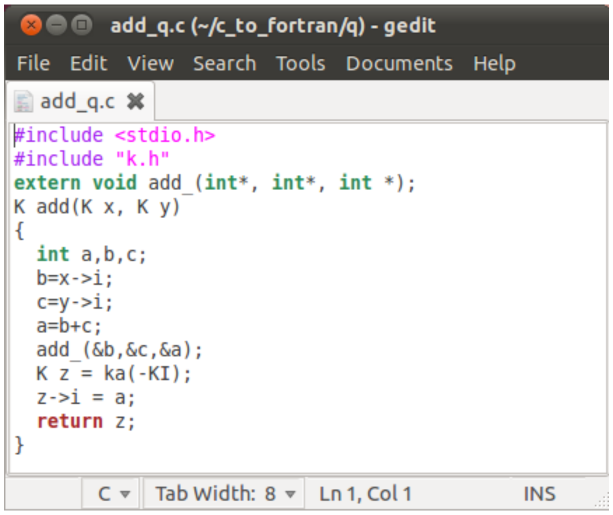

johnl@johnl-VirtualBox: ~
File Edit View Search Terminal Help

```
bash 4.1$ fort77 -c add.f
   add:
bash 4.1$ gcc -o add.x add_two_numbers.c add.o
bash 4.1$ ./add.x
5
bash 4.1$
```

3 Calling the C wrapper from kdb+

- For information on interfacing C with kdb+, see:
  https://code.kx.com/trac/wiki/Cookbook/InterfacingWithC
  and
  https://code.kx.com/trac/wiki/Cookbook/ExtendingWithC
  The header file k.h needs to be included in your C code in order to create and access K objects. k.h can be downloaded from:
  https://code.kx.com/trac/browser/kx/kdb%2B/c/c/k.h

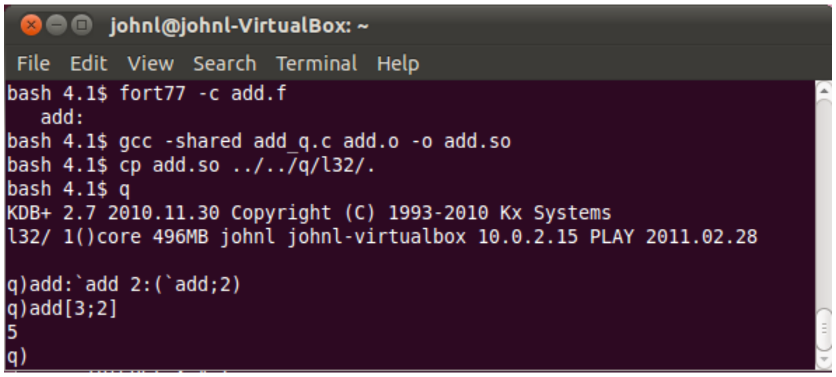
- Now we modify the add_two_numbers.c code to accept input and return output from kdb+. The code add_q.c is shown below:


add_q.c (~/c_to_fortran/q) - gedit
File Edit View Search Tools Documents Help
add_q.c
C
Tab Width: 8
Ln 1, Col 1
INS

```
#include <stdio.h>
#include "k.h"
extern void add_(int*, int*, int *);
K add(K x, K y)
{
  int a,b,c;
  b=x->i;
  c=y->i;
  a=b+c;
  add_(&b,&c,&a);
  K z = ka(-KI);
  z->i = a;
  return z;
}
```

Two K objects of type integer are read in as input and one K object of type integer is created as ouput and returned to kdb+.

- Now we compile the Fortran code add.f and link the object file to the C code add_q.c to form a shared object add.so. The shared object add.so is then placed in the $QHOME/l32 directory. In q, the shared object library can then be dynamically loaded using 2:


johnl@johnl-VirtualBox: ~
File Edit View Search Terminal Help

```
bash 4.1$ fort77 -c add.f
   add:
bash 4.1$ gcc -shared add_q.c add.o -o add.so
bash 4.1$ cp add.so ../../q/l32/.
bash 4.1$ q
KDB+ 2.7 2010.11.30 Copyright (C) 1993-2010 Kx Systems
l32/ 1()core 496MB johnl johnl-virtualbox 10.0.2.15 PLAY 2011.02.28

q)add:`add 2:(`add;2)
q)add[3;2]
5
q)
```

4 Calling a Fast Fourier Transform (FFT) from kdb+

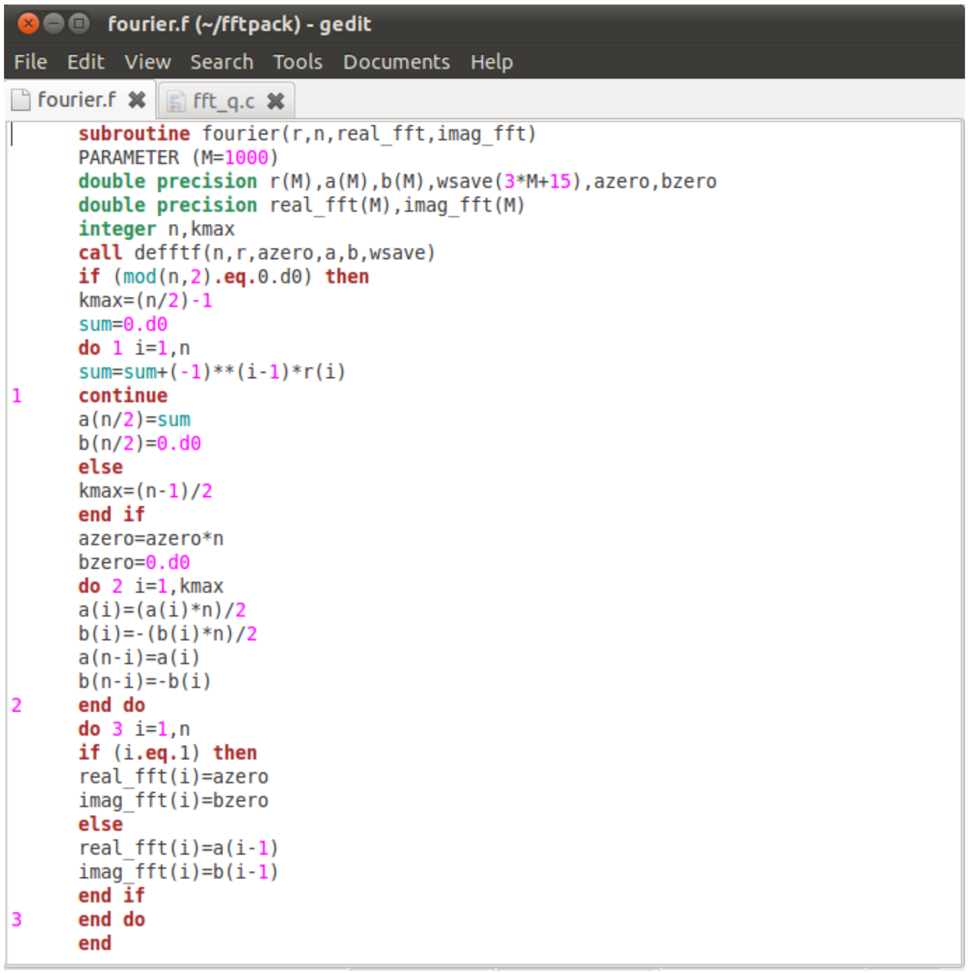
- Now we move onto a more realistic example. In signal processing, FFT's are often used to decompose a signal into its component frequencies. A popular package for performing FFT's is the FFTPACK package, originally written in Fortran 77. The Fortran subroutine defftf.f calculates the fourier coefficients of a real periodic sequence. Here we use the double precision version of this subroutine (and its associated dependencies) available at:
  http://www.netlib.org/cgi-bin/netlibfiles.pl?filename=/bihar/defftf.f
  We will now show how to call the fortran subroutine defftf.f from q.

- There are various conventions in use regarding the form of the discrete fourier transform, see:
  http://reference.wolfram.com/mathematica/tutorial/FourierTransforms.html

Here, we will choose the signal processing convention where the discrete fourier transform is of the form:

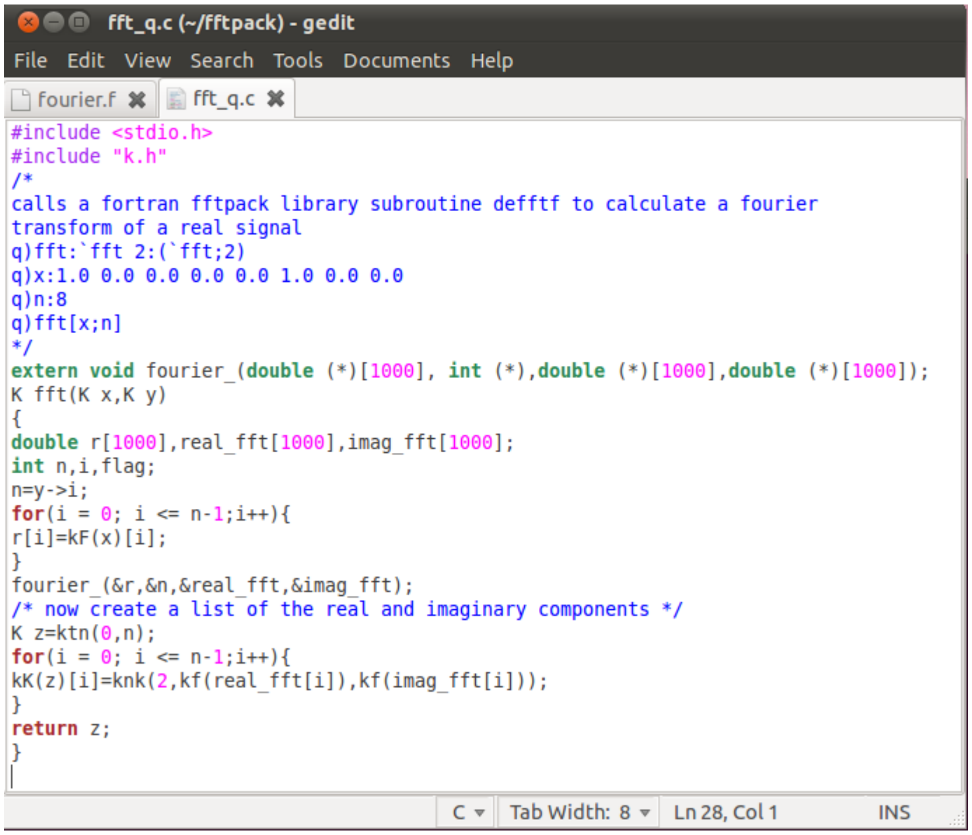
$$v_s = \sum_{r=1}^{n} u_r \exp(-2\pi i(r-1)(s-1)/n)$$

where u_r is the n element signal to be transformed. The Fortran subroutine fourier.f calls the defftf.f FFTPACK routine and converts the output to this convention:


fourier.f (~/fftpack) - gedit
File Edit View Search Tools Documents Help
fourier.f
fft_q.c

```
      subroutine fourier(r,n,real_fft,imag_fft)
      PARAMETER (M=1000)
      double precision r(M),a(M),b(M),wsave(3*M+15),azero,bzero
      double precision real_fft(M),imag_fft(M)
      integer n,kmax
      call defftf(n,r,azero,a,b,wsave)
      if (mod(n,2).eq.0.d0) then
      kmax=(n/2)-1
      sum=0.d0
      do 1 i=1,n
      sum=sum+(-1)**(i-1)*r(i)
1     continue
      a(n/2)=sum
      b(n/2)=0.d0
      else
      kmax=(n-1)/2
      end if
      azero=azero*n
      bzero=0.d0
      do 2 i=1,kmax
      a(i)=(a(i)*n)/2
      b(i)=-(b(i)*n)/2
      a(n-i)=a(i)
      b(n-i)=-b(i)
2     end do
      do 3 i=1,n
      if (i.eq.1) then
      real_fft(i)=azero
      imag_fft(i)=bzero
      else
      real_fft(i)=a(i-1)
      imag_fft(i)=b(i-1)
      end if
3     end do
      end
```

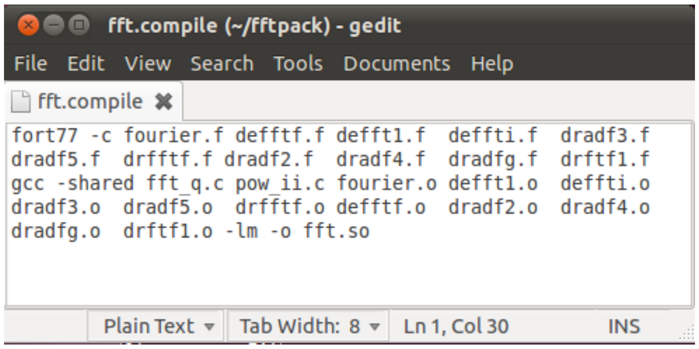
- Now we write the C wrapper to this routine fft_q.c. This calls the fourier.f fortran routine


fft_q.c (~/fftpack) - gedit
File Edit View Search Tools Documents Help
fourier.f
fft_q.c
C
Tab Width: 8
Ln 28, Col 1
INS

```c
#include <stdio.h>
#include "k.h"
/*
calls a fortran fftpack library subroutine defftf to calculate a fourier
transform of a real signal
q)fft:`fft 2:(`fft;2)
q)x:1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
q)n:8
q)fft[x;n]
*/
extern void fourier_(double (*)[1000], int (*),double (*)[1000],double (*)[1000]);
K fft(K x,K y)
{
double r[1000],real_fft[1000],imag_fft[1000];
int n,i,flag;
n=y->i;
for(i = 0; i <= n-1;i++){
r[i]=kF(x)[i];
}
fourier_(&r,&n,&real_fft,&imag_fft);
/* now create a list of the real and imaginary components */
K z=ktn(0,n);
for(i = 0; i <= n-1;i++){
kK(z)[i]=knk(2,kf(real_fft[i]),kf(imag_fft[i]));
}
return z;
}
```

fft_q.c takes as input from q the list of floats to be transformed (x) and the number of elements in the list (y). Individual elements in x are accessed in C via the kF(x) object accessor. See:
https://code.kx.com/trac/wiki/Cookbook/InterfacingWithC#ExaminingKObjects
The output to the FFT is a list of complex numbers. This will be represented in q by a list, each of whose elements is itself a two-element list containing the real and imaginary components. To construct this mixed list in C we make use of the ktn function to construct the K object z. Individual elements of this mixed list are accesssed via the kK(z) function, with the two item mixed list then created by the knk function, and each item with the kf function. See:
https://code.kx.com/trac/wiki/Cookbook/InterfacingWithC#Creatinglists

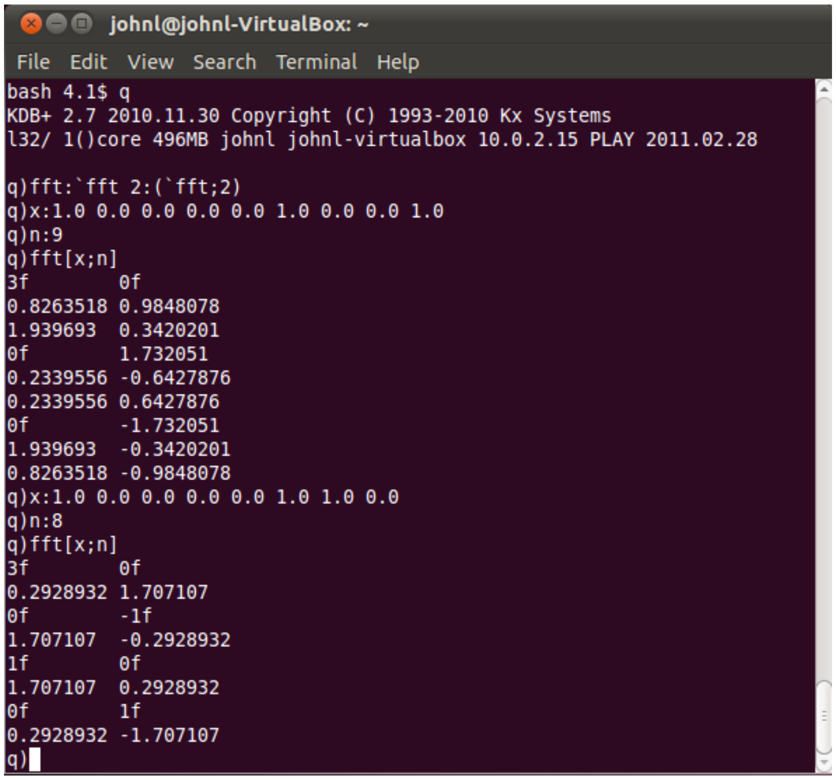
- As for the simpler example of the addition of two numbers discussed earlier, the Fortran routines are first compiled, then the resulting object files linked to the C code. Note that a link to the C math library is required usng the linker option -lm. In addition the C routine pow_ii.c available at:
  http://www.netlib.org/templates/single/pow_ii.c
  is required. The compilation line is of the form:

  
  fft.compile (~/fftpack) - gedit
File Edit View Search Tools Documents Help
fft.compile
Plain Text
Tab Width: 8
Ln 1, Col 30
INS

```
fort77 -c fourier.f defftf.f defft1.f  deffti.f  dradf3.f
dradf5.f  drfftf.f dradf2.f  dradf4.f  dradfg.f  drftf1.f
gcc -shared fft_q.c pow_ii.c fourier.o defft1.o  deffti.o
dradf3.o  dradf5.o  drfftf.o defftf.o  dradf2.o  dradf4.o
dradfg.o  drftf1.o -lm -o fft.so
```

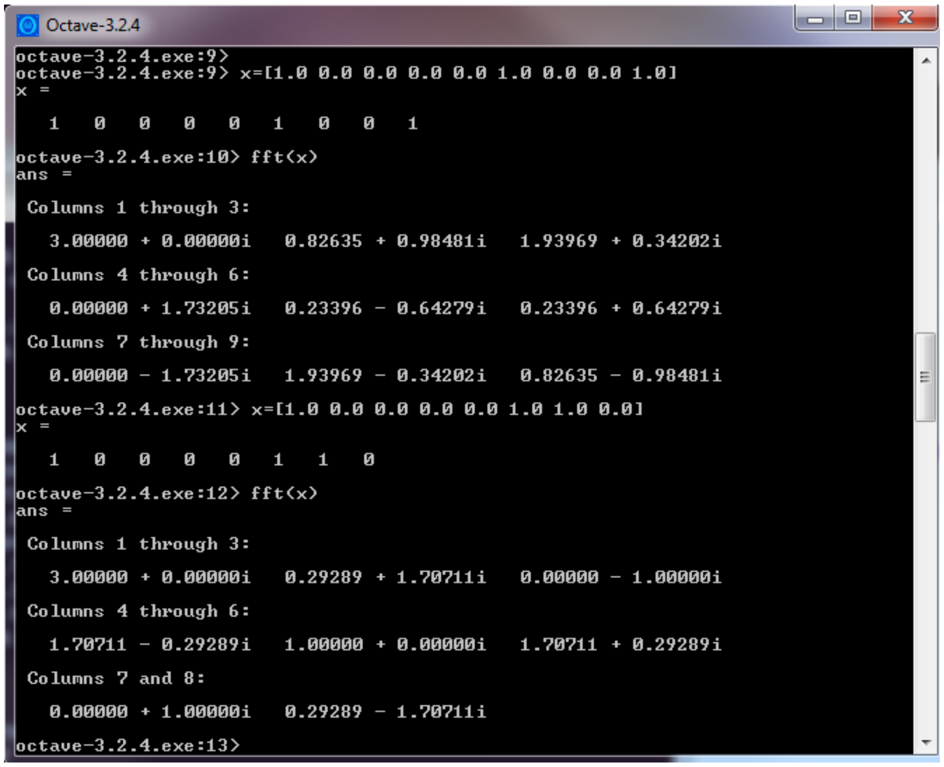
  The shared object fft.so is then placed in the $QHOME/l32 directory and can then be loaded into q using 2:

- The subroutine defftf.f is not restricted to a power of 2 for the number of elements in the list, and can be called for a general list of floats. Here we show the result of taking the FFT of an 8, and a 9 element list of floats that are defined in q:


johnl@johnl-VirtualBox: ~
File Edit View Search Terminal Help

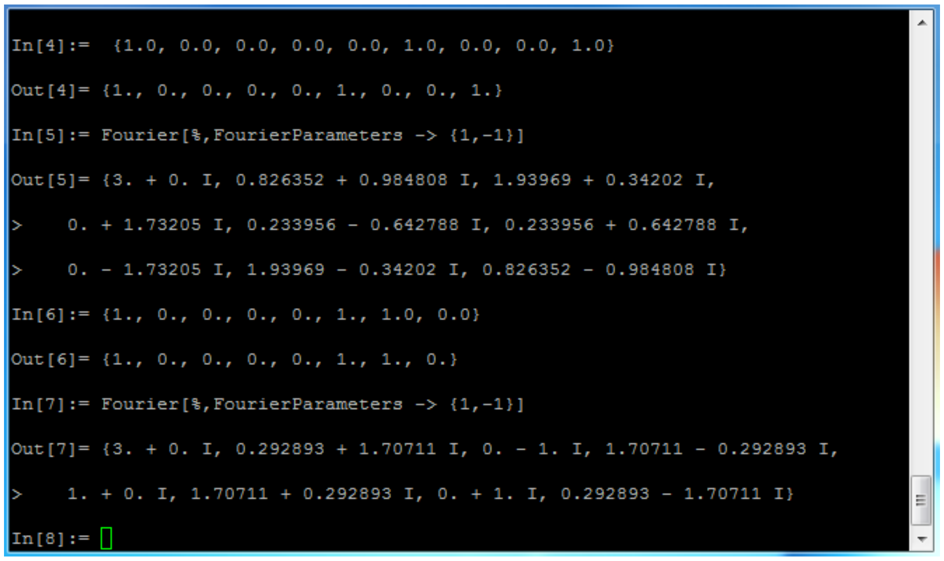
```
bash 4.1$ q
KDB+ 2.7 2010.11.30 Copyright (C) 1993-2010 Kx Systems
l32/ 1()core 496MB johnl johnl-virtualbox 10.0.2.15 PLAY 2011.02.28

q)fft:`fft 2:(`fft;2)
q)x:1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0
q)n:9
q)fft[x;n]
3f         0f
0.8263518 0.9848078
1.939693  0.3420201
0f         1.732051
0.2339556 -0.6427876
0.2339556 0.6427876
0f         -1.732051
1.939693  -0.3420201
0.8263518 -0.9848078
q)x:1.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0
q)n:8
q)fft[x;n]
3f         0f
0.2928932 1.707107
0f         -1f
1.707107  -0.2928932
1f         0f
1.707107  0.2928932
0f         1f
0.2928932 -1.707107
q)
```

- For comparison and testing purposes, we show the same fourier transforms in Octave:



```
octave-3.2.4.exe:9>
octave-3.2.4.exe:9> x=[1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0]
x =

   1   0   0   0   0   1   0   0   1

octave-3.2.4.exe:10> fft(x)
ans =

 Columns 1 through 3:

   3.00000 + 0.00000i   0.82635 + 0.98481i   1.93969 + 0.34202i

 Columns 4 through 6:

   0.00000 + 1.73205i   0.23396 - 0.64279i   0.23396 + 0.64279i

 Columns 7 through 9:

   0.00000 - 1.73205i   1.93969 - 0.34202i   0.82635 - 0.98481i

octave-3.2.4.exe:11> x=[1.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0]
x =

   1   0   0   0   0   1   1   0

octave-3.2.4.exe:12> fft(x)
ans =

 Columns 1 through 3:

   3.00000 + 0.00000i   0.29289 + 1.70711i   0.00000 - 1.00000i

 Columns 4 through 6:

   1.70711 - 0.29289i   1.00000 + 0.00000i   1.70711 + 0.29289i

 Columns 7 and 8:

   0.00000 + 1.00000i   0.29289 - 1.70711i

octave-3.2.4.exe:13>
```

and in Mathematica:



```
In[4]:=  {1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0}

Out[4]= {1., 0., 0., 0., 0., 1., 0., 0., 1.}

In[5]:= Fourier[%,FourierParameters -> {1,-1}]

Out[5]= {3. + 0. I, 0.826352 + 0.984808 I, 1.93969 + 0.34202 I,

>    0. + 1.73205 I, 0.233956 - 0.642788 I, 0.233956 + 0.642788 I,

>    0. - 1.73205 I, 1.93969 - 0.34202 I, 0.826352 - 0.984808 I}

In[6]:= {1., 0., 0., 0., 0., 1., 1.0, 0.0}

Out[6]= {1., 0., 0., 0., 0., 1., 1., 0.}

In[7]:= Fourier[%,FourierParameters -> {1,-1}]

Out[7]= {3. + 0. I, 0.292893 + 1.70711 I, 0. - 1. I, 1.70711 - 0.292893 I,

>    1. + 0. I, 1.70711 + 0.292893 I, 0. + 1. I, 0.292893 - 1.70711 I}

In[8]:=
```

5 Comments

This report should provide some initial guidance about how to go about making a fortran subroutine callable from q. Some further topics such as passing character strings from q to Fortran via C have been omitted. More guidance on interfacing C and Fortran programs can be found at:

- http://www.nersc.gov/nusers/resources/software/ibm/c_and_f.php
- http://www.ibiblio.org/pub/languages/fortran/ch1-11.html
- http://idlastro.gsfc.nasa.gov/idl_html_help/Fortran_Examples.html

See also the Q math library available at
http://althenia.net/qml
which provides a collection of various mathematical routines in C that are callable from kdb+.