# Stay at CHMU from 29th October to 8th November 2013

Juan Simarro, AEMET, Spain

November 18, 2013

## 1 VFE implementation

The main subject of the stay was the VFE implementation developed by Josef Vivoda and Petra Smolíková. There is a purpose of writing a paper about this issue in a scientific review, based on the HIRLAM Newsletter number 60, August 2013, *Finite elements used in the vertical discretization of the fully compressible forecast model ALADIN-NH* by Josef Vivoda and Petra Smolíková.

Some issues were discussed and worked out.

First, as we were advised, in order to write the paper is should be demonstrated that the VFE scheme improves the forecast when compared with the operational VFD scheme. This task seems not to be easy, as the improvement in the accuracy of the vertical operators is somewhat masked with other sources of error, as physical parametrization and time stepping. Comparison of analytical and numerical normal modes, without time discretization and for a fixed horizontal wave number, could help to demonstrate the higher accuracy of the VFE. However, this exercise was not possible, as we do not have analytical normal modes of the whole linear system in the mass based vertical coordinate. Linear two dimensional dry tests could help in demonstrating the higher accuracy of the VFE scheme. To this end, an analytical linear solution of the Euler equations over non flat orography must be constructed. An extension of the work of *An analytic solution for linear gravity waves in a channel as a test for numerical models using the non-hydrostatic compressible Euler equations* by Michael Baldauf and Slavko Brdar could help to this end.

Another point of interest was the transformation from vertical velocity to vertical divergence. This part of the vertical discretization was still first order and then a bottle neck which should be solved. It is necessary for this purpose to dispose of a set of vertical integral and derivative operators that are exactly invertible. Josef Vivoda and Alvaro Subías solved this problem independently and so I did. The solution are similar in the sense that all are based on the fact that the integral of a B-spline of order $k$ is a B-spline of order $k+1$ and the derivative of a B-spline of order $k+1$ is a B-spline of order $k$, because the integral operator increases in one the continuity of the function and the derivative decreases it in one. In this report there is a full explanation of the method I have found with convergence tests. There are two points that are not satisfactory in the results. Firstly, the derivative-integral condition is not exactly fulfilled although the error is constant when applied repeatedly the derivative-integral transformations. Secondly, the error in the derivative operator has

an oscillatory pattern, although it satisfies convergence for $L_\infty$ and $L_2$ norms. Whether this facts can affect the full non linear model behavior is not an easy question to answer.

A third point of interest was why boundary conditions in the vertical discretization are so important for the linear and non linear model stability. Josef Vivoda has got a VFE stable model for some choice of the vertical laplacian, integral operators and boundary conditions. However, from my point of view it will be interesting to better understand, if it is possible, the relations between the vertical laplacian boundary conditions and the stability. Also, which are the boundary conditions for the integral operators that are more appropriate for satisfying the constraints of the linear system.

In conclusion, I have the feeling that the VFE scheme is mature and a non linear stable model have been implemented. However, there are some open questions that could be answered to try understand some important points. The answers of this questions could help to publish the work, as the scientific community like to know not only that a VFE implementation works but why. For this objective, a more systematic study of the impact of boundary conditions in the stability of the model could help. I will continue the work on this topic during next weeks. The conclusions, if any, will be reported in a separate document.

## 2 Integral and derivative operators

The derivative and integral operators in this section are

$$D(f)(z) = \frac{\partial f}{\partial z}(z)$$
$$Q(f)(z) = \int_0^z f(y)dy$$

The invertibility conditions for this operators are

$$(D \circ Q)(f) = f$$
$$(Q \circ D)(f) = f - f(0)$$

Then we seek for three matrices $\mathbf{D}$, $\mathbf{Q}$ and $\mathbf{E}$ such that, when applied to a discretized function $\mathbf{f}$, verify

$$\mathbf{D} \cdot \mathbf{Q} \cdot \mathbf{f} = \mathbf{f}$$
$$\mathbf{Q} \cdot \mathbf{D} \cdot \mathbf{f} = \mathbf{f} - \mathbf{E} \cdot \mathbf{f}$$

where $\mathbf{E} \cdot \mathbf{f}$ is an extrapolated value of function $\mathbf{f}$ at $z = 0$. Here the function $\mathbf{f}$ is given at full levels without boundary conditions, therefore value of function $\mathbf{f}$ at $z = 0$ must be extrapolated numerically.

The method described here can be modified to include boundary conditions. In that case, boundary conditions should be consistent in the sense that, for instance, if $f$ is zero at boundaries then the $Q(f)$ must have first derivative zero at boundaries.

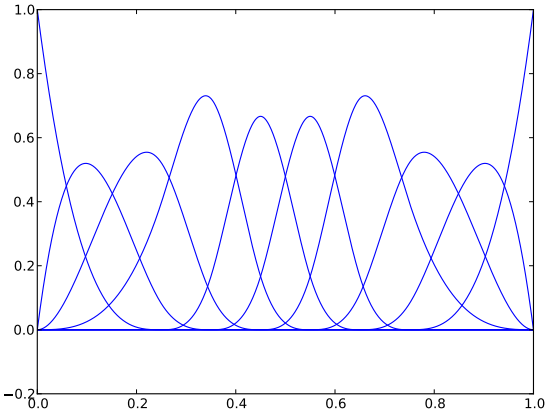


Figure 1: B-spline basis for $N = 10$ full levels regularly spaced.

## 2.1 Full levels and B-spline representation

Full levels are a set of $N$ values $\{z_i\}$ between 0 and 1. In the test shown bellow full levels are regularly spaced, although they can be placed in any convenient set of full levels. The B-splines knots are calculated from full levels. The number of knots is $K = N + B - C$ being $N$ the number of full levels, $B$ the number of boundary conditions and $C$ the order of the splines. There are not boundary conditions ($B = 0$) and the B-splines used in the test are linear, cubic or quintic ($C = 1, 3, 5$). In figure 1 are plotted the B-spline basis for $N = 10$ and $C = 3$. The B-spline basis is a set of $N$ piecewise polynomials $\{S_i(z)\}$. The matrix **S2L** transforms from B-spline space to full levels and it is defined as

$$(S2L)_{ij} = S_j(z_i) \text{ for } i, j = 1, \ldots, N$$

The inverse $\mathbf{L2S} = \mathbf{S2L}^{-1}$ transforms from full levels to B-spline space. The interpolated value of $\mathbf{f}$ at $z = 0$, which is necessary for the derivative operator, is

$$\mathbf{f_0} = \mathbf{E} \cdot \mathbf{f}$$

There are many ways of defining the extrapolating matrix $\mathbf{E}$. The one used here is to extrapolate the function using the B-spline representation given by **L2S**.

## 2.2 Integral operator

The construction of the integral operator $\mathbf{Q}$ is as follow

$$\mathbf{Q} = \hat{\mathbf{Q}} \cdot \mathbf{L2S}$$

where the matrix $\hat{\mathbf{Q}}$ is

$$(\hat{Q})_{ij} = \int_0^{z_i} S_j(y)dy \text{ for } i, j = 1, \ldots, N$$

## 2.3 Derivative operator

The construction of the derivative operator $\mathbf{D}$ is as follow

$$\mathbf{D} = \mathbf{Q}^{-1} \cdot (\mathbf{I} - \mathbf{E})$$

where the matrix $\mathbf{I}$ is the identity and matrix $\mathbf{E}$ gives the extrapolated value at $z = 0$. The reason to include the $\mathbf{E}$ operator is to remove the constant function $\mathbf{f_0}$ consiting of the value of the function at $z = 0$, because $\mathbf{Q}^{-1}$ must be applied to functions which are zero at $z = 0$ as $\mathbf{Q}$ returns always zero at $z = 0$.

## 2.4 Invertibility conditions

The invertibility conditions are

$$\mathbf{D} \cdot \mathbf{Q} = \mathbf{Q}^{-1} \cdot (\mathbf{I} - \mathbf{E}) \cdot \mathbf{Q} = \mathbf{I} - \mathbf{Q}^{-1} \cdot \mathbf{E} \cdot \mathbf{Q}$$
$$\mathbf{Q} \cdot \mathbf{D} = \mathbf{Q} \cdot \mathbf{Q}^{-1} \cdot (\mathbf{I} - \mathbf{E}) = \mathbf{I} - \mathbf{E}$$

The first condition is fulfilled if $\mathbf{E} \cdot \mathbf{Q} \cdot \mathbf{f} = \mathbf{0}$. Although this is true analytically, because the definite integral of any continuous function from $z = 0$ is zero at $z = 0$, it is not in the discretization. However, there are two facts to be considered. One is that the error does not grow when applying many times the derivative and integral operators. This is easily shown taking into account that by construction operator $\mathbf{E}$ is idempotent, that is $\mathbf{E}^n = \mathbf{E}$. As a consequence $\mathbf{I} - \mathbf{E}$ is idempotent as well and therefore

$$(\mathbf{Q} \cdot \mathbf{D})^n = (\mathbf{I} - \mathbf{E})^n = \mathbf{I} - \mathbf{E}$$

On the other hand

$$(\mathbf{D} \cdot \mathbf{Q})^n = \mathbf{Q}^{-1} \cdot (\mathbf{I} - \mathbf{E})^n \cdot \mathbf{Q} = \mathbf{I} - \mathbf{Q}^{-1} \cdot \mathbf{E} \cdot \mathbf{Q}$$

Therefore, although $\mathbf{D} \cdot \mathbf{Q} \neq \mathbf{I}$, the error does not grow when applying repeatedly the operators and it is always $\mathbf{Q}^{-1} \cdot \mathbf{E} \cdot \mathbf{Q}$. Moreover, the eigenvalues of $\mathbf{D} \cdot \mathbf{Q}$ have module 1 with very small imaginary part and real part near 1, except one which is 0. This is consistent with the analytical counterpart of these operators.

## 2.5 Test

The test function used in the tests reported in this section is

$$f(z) = \frac{30z^5 - 19z^4 + 55z^3 + 12z^2 - 65z + 21}{30z^4 + 60z^2 + 30} e^z$$

The integral and derivative are

$$D(f)(z) = \frac{30z^7 + 11z^6 + 85z^5 + 88z^4 - 110z^3 + 393z^2 - 125zp - 44}{30z^6 + 90z^4 + 90z^2 + 30.0} e^z$$
$$I(f)(z) = \frac{30z^3 - 49z^2 + 25z - 4}{30z^2 + 30} e^z + \frac{4}{30}$$

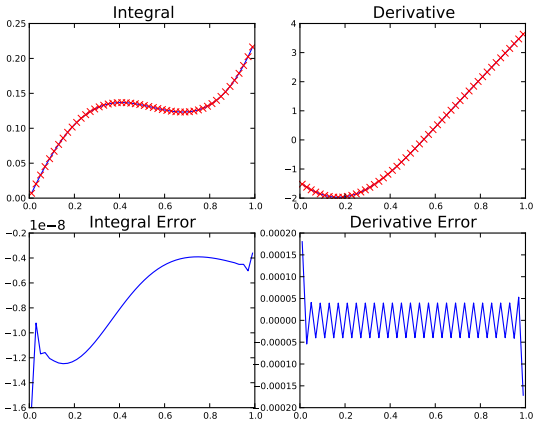


Figure 2: Derivative $\mathbf{D}$ and integral $\mathbf{Q}$ operators applied to the test function, for cubic B-spline and 50 full levels. Above, analytical values in blue, numerical values in red crosses. Bellow, the difference between numerical and analytical values

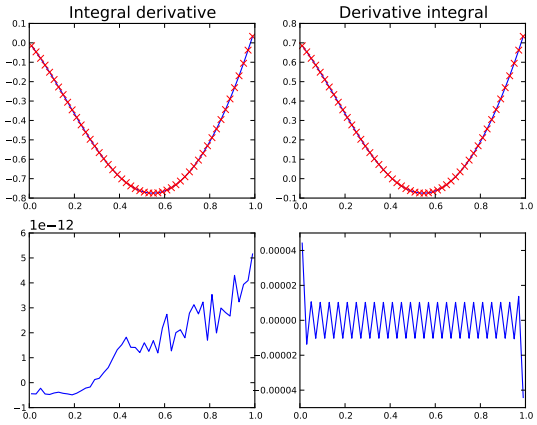


Figure 3: Upper left plot, $\mathbf{f} - \mathbf{E} \cdot \mathbf{f}$ in blue and $(\mathbf{Q} \cdot \mathbf{D})^{300} \cdot \mathbf{f}$ in red crosses. Upper right plot, $\mathbf{f}$ in blue and $(\mathbf{D} \cdot \mathbf{Q})^{300} \cdot \mathbf{f}$ in red crosses. Lower left plot $(\mathbf{Q} \cdot \mathbf{D})^{300} \cdot \mathbf{f} - (\mathbf{f} - \mathbf{E} \cdot \mathbf{f})$. Lower right plot, $(\mathbf{D} \cdot \mathbf{Q})^{300} \cdot \mathbf{f} - \mathbf{f}$.

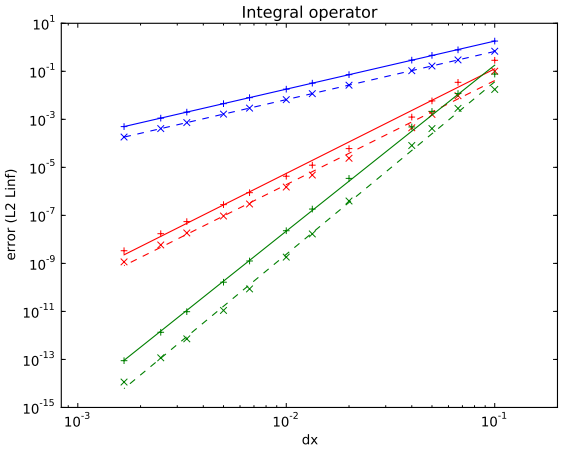


Figure 4: Convergence of integral operator **Q**. Blue, red and green colors correspond to linear, cubic and quintic B-splines representation respectively. Continuous and dashed lines correspond to a linear logarithmic regression between the $L_\infty$ and $L_2$ errors and the grid spacing. The crosses are the errors when considering a number of levels equal to 10, 15, 20, 25, 50, 75, 100, 150, 200, 300, 400 and 600.

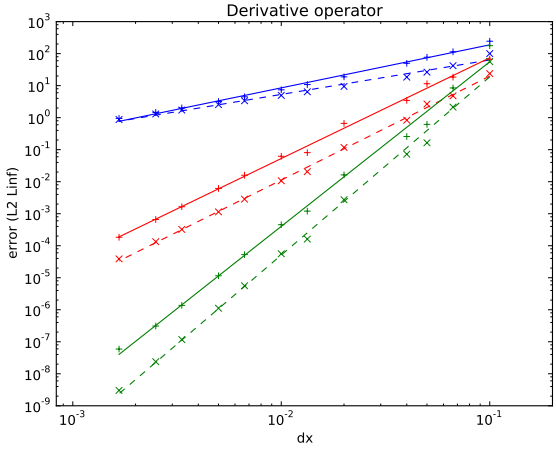


Figure 5: Convergence of integral operator **D**. Blue, red and green colors correspond to linear, cubic and quintic B-splines representation respectively. Continuous and dashed lines correspond to a linear logarithmic regression between the $L_\infty$ and $L_2$ errors and the grid spacing. The crosses are the errors when considering a number of levels equal to 10, 15, 20, 25, 50, 75, 100, 150, 200, 300, 400 and 600.

The following test functions have been also considered

$$
\begin{aligned}
f(z) &= \sin^3 z \cos z \\
f(z) &= b\cos(az)e^{bz} - a\sin(az)e^{bz} \\
f(z) &= z^n \\
f(z) &= \sin(k\pi z) \\
f(z) &= \cos(k\pi z) \\
f(z) &= 1
\end{aligned}
$$

The **D** and **I** operators are applied to the test function using B-spline of order $C$ equal 1, 3 and 5 and number of levels $N$ equal 10, 15, 20, 25, 50, 75, 100, 150, 200, 300, 400 and 600. The results in figure 2 are for the case C=3 and N=50. It is obvious that the error of the derivative has an oscillatory pattern which is not satisfactory, although the error is bounded to the convergence values, as it is shown later.

The invertibility properties for this test case are shown in figure 3.

$$
\begin{aligned}
(\mathbf{Q}\cdot\mathbf{D})^{300}\cdot\mathbf{f} &= \mathbf{f} - \mathbf{E}\cdot\mathbf{f} \\
(\mathbf{D}\cdot\mathbf{Q})^{300}\cdot\mathbf{f} &\simeq \mathbf{f}
\end{aligned}
$$

The first condition is fulfilled exactly, up to round-off error. The second condition is not fulfilled, although it is constant as shown above and equal to $\mathbf{Q}^{-1}\cdot\mathbf{E}\cdot\mathbf{Q}\cdot\mathbf{f}$.

## 2.6 Convergence

The convergence of the operators are shown in figures 4 and 5. It is clear that the errors of the integral are smaller than the errors of the derivative, although the convergence, given by the slope of the linear logarithmic regression between errors and grid spacing, are similar. The slopes are

| C | $L_\infty(\mathbf{D})$ | $L_2(\mathbf{D})$ | $L_\infty(\mathbf{Q})$ | $L_2(\mathbf{Q})$ |
|---|---|---|---|---|
| 1 | 1.344 | 1.075 | 2.002 | 2.006 |
| 3 | 3.144 | 3.246 | 4.356 | 4.335 |
| 5 | 5.139 | 5.565 | 6.914 | 7.185 |

For cubic splines the derivative converges at a rate of 3.1 and the integral at 4.3.

## 2.7 Python program

The `python` program is listed bellow. The `scipy.interpolate` package is used for finding the B-splines. This package is able to use B-splines up to quintic order. In particular, it is used the class `LSQUnivariateSpline` and the methods `integrate` and `derivatives`. The `linalg` package is used for matrix inversion and eigenvalues.

The routine `find_bspline_basis` finds the B-spline space given the full levels and the top and bottom. The routine `find_operators_D_and_I` constructs the derivative and integral operators. The routine `plot_test` does and plots the tests.

```
import numpy as np
import matplotlib.pyplot as pt

# ----------------------------------------
# Integral and derivative operators at full levels of
# a function without boundary conditions

def levels(zn,z0,z1):
    yp = np.linspace(z0,z1,zn+1)
    zp = yp[0:zn]+0.5*yp[1]
    return zp

def find_bspline_basis(zn,z0,z1,zp):
    # interpolate package
    import scipy.interpolate as it
    # nodes
    xp = zp.copy();
    xn = zn; x0 = z0; x1 = z1; xp[0] = x0; xp[zn-1] = x1
    # internal knots depending on spline order
    kp = xp[int((kc+1)/2):-int((kc+1)/2)]
    # matrix from level to spline
    l2s = np.zeros([xn,xn])
    # find l2s
    for i in range(0,xn):
        # value of the function at model levels
        yp = np.zeros([xn]); yp[i] = 1.0
        # spline interpolation
        sp = it.LSQUnivariateSpline(xp,yp,kp,k=kc)
        # get coefficients
        l2s[:,i] = sp.get_coeffs()
    # find s2l
    s2l = np.linalg.inv(l2s)
    # find set of splines
    sps = []
    for i in range(0,xn):
        # value of the function at model levels
        yp = s2l[:,i]
        # spline interpolation
        sp = it.LSQUnivariateSpline(xp,yp,kp,k=kc)
        # add to set
        sps.append(sp)
    # plot set of splines
    plot_splines(xn,x0,x1,sps)
    # find set of splines from s2l
    return sps

def plot_splines(xn,x0,x1,sps):
    f,axarr = pt.subplots(1,1)
    xs = np.linspace(x0,x1,1000)
    for i in range(0,xn):
        ys = sps[i](xs)
        axarr.plot(xs,ys,'b-')
    f.savefig('zn_%03d_splines.pdf' % xn,format='pdf')
    pt.close()

def find_operators_D_and_I(kc,zn):
    # kc: spline order (3 cubic)
    # zn: number of full levels and domain
    z0 = 0.0; z1 = 1.0
    # model levels
    zp = levels(zn,z0,z1)
    # b spline basis
    sps = find_bspline_basis(zn,z0,z1,zp)
    # Spline to level space without boundary conditions
    S2L_op = np.zeros([zn,zn])
    for i in range(0,zn):
        for j in range(0,zn):
            # values of the splines at model levels
            S2L_op[j,i] = sps[i].derivatives(zp[j])[0]
    # Level to spline space without boundary conditions
    L2S_op = np.linalg.inv(S2L_op)
    # Integral operator
    T_op = np.zeros([zn,zn])
```

```
    for i in range(0,zn):
        # find definite integral on model levels
        for j in range(0,zn):
            T_op[j,i] = sps[i].integral(z0,zp[j])
    Q_op = np.dot(T_op,L2S_op)
    # How is it
    print np.linalg.cond(Q_op)
    print np.linalg.det(Q_op)
    # constant function
    I_op = np.eye(zn)
    E_op = np.zeros([zn,zn])
    for i in range(0,zn):
        E_op[i,:] = L2S_op[0,:]
    # Derivative operator
    D_op = np.dot(np.linalg.inv(Q_op),I_op-E_op)
    # return
    return z0,zp,D_op,Q_op,E_op

def plot_test(zn,zp,fp,IA_fp,DA_fp,Q_op,D_op,E_op,name):
    # Four axes, returned as a 2-d array
    f,axarr = pt.subplots(2,2)
    # integral
    IN_fp = np.dot(Q_op,fp)
    axarr[0,0].plot(zp,IA_fp,'b-')
    axarr[0,0].plot(zp,IN_fp,'rx')
    axarr[0,0].set_title('Integral')
    axarr[1,0].plot(zp,IN_fp-IA_fp,'b-')
    axarr[1,0].set_title('Integral Error')
    # derivative
    DN_fp = np.dot(D_op,fp)
    axarr[0,1].plot(zp,DA_fp,'b-')
    axarr[0,1].plot(zp,DN_fp,'rx')
    axarr[0,1].set_title('Derivative')
    axarr[1,1].plot(zp,DN_fp-DA_fp,'b-')
    axarr[1,1].set_title('Derivative Error')
    f.savefig(name+'_I_and_D.pdf',format='pdf')
    pt.close()
    # Four axes, returned as a 2-d array
    f,axarr = pt.subplots(2,2)
    # integral derivative
    QDN_fp = np.dot(np.linalg.matrix_power(np.dot(Q_op,D_op),300),fp)
    axarr[0,0].plot(zp,fp-np.dot(E_op,fp),'b-')
    axarr[0,0].plot(zp,QDN_fp,'rx')
    axarr[0,0].set_title('Integral derivative')
    axarr[1,0].plot(zp,QDN_fp-(fp-np.dot(E_op,fp)),'b-')
    print 'max abs QDN_fp-fp: %20.10e' % np.max(np.abs(QDN_fp-fp))
    # derivative integral
    DQN_fp = np.dot(np.linalg.matrix_power(np.dot(D_op,Q_op),300),fp)
    axarr[0,1].plot(zp,fp,'b-')
    axarr[0,1].plot(zp,DQN_fp,'rx')
    axarr[0,1].set_title('Derivative integral')
    axarr[1,1].plot(zp,DQN_fp-fp,'b-')
    print 'max abs DQN_fp-fp: %20.10e' % np.max(np.abs(DQN_fp-fp))
    f.savefig(name+'_ID_and_DI.pdf',format='pdf')
    pt.close()
    # Print errors
    w = open(name+'_errors.txt','w')
    w.write('D Linf L2: %20.10e %20.10e\n' % (np.max(np.abs(DN_fp-DA_fp)),np.std(DN_fp-DA_fp)) )
    w.write('I Linf L2: %20.10e %20.10e\n' % (np.max(np.abs(IN_fp-IA_fp)),np.std(IN_fp-IA_fp)) )
    w.close

def make_test(kc,zn,zp,Q_op,D_op,E_op,z0):

    # ----------------------------------------
    # Test exp cos
    ka = 11.0; kb = 3.0
    fp = (kb*np.cos(ka*zp)-ka*np.sin(ka*zp))*np.exp(kb*zp)
    IA_fp = np.cos(ka*zp)*np.exp(kb*zp)-1.0
    DA_fp = ((kb**2-ka**2)*np.cos(ka*zp)-2.0*ka*kb*np.sin(ka*zp))*np.exp(kb*zp)
    plot_test(zn,zp,fp,IA_fp,DA_fp,Q_op,D_op,E_op,'zn_%03d_kc_%1d_expcos' % (zn,kc) )

    # ----------------------------------------
    # Test cos function
    ka = 11; kb = ka*np.pi
```

```
    fp = np.cos(kb*zp)
    IA_fp = +(np.sin(kb*zp)-np.sin(kb*z0))/kb
    DA_fp = -np.sin(kb*zp)*kb
    plot_test(zn,zp,fp,IA_fp,DA_fp,Q_op,D_op,E_op,'zn_%03d_kc_%1d_cos11' % (zn,kc) )

for kc in [1,3,5]:
    for zn in [10,15,20,25,50,75,100,150,200,300,400,600]:
        z0,zp,D_op,Q_op,E_op = find_operators_D_and_I(kc,zn)
        make_test(kc,zn,zp,Q_op,D_op,E_op,z0)
```