`FunFlows`: Functional Renormalization Group for O(1) Scalar Field Theory in d-dimensions

Tony Menzo

Abstract

`FunFlows` is a symbolic and numerical software library that implements functional renormalization group methods to compute the beta-functions and fixed-points of a $O(N)$-symmetric scalar field theories in d-dimensions.

1 Functional Renormalization Group

In quantum field theories all physical content is stored in n-point correlation functions

$$\langle \phi_i(x_1)\phi_j(x_2)\cdots\phi_k(x_n)\rangle = \frac{1}{Z[0]}\left[\frac{\delta^n Z[J]}{\delta J_i(x_1)\delta J_j(x_2)\cdots\delta J_k(x_n)}\right]_{J=0} \tag{1.0.1}$$

Where $Z[J]$ is a generating functional of correlation functions. The core entity in functional renormalization group is the ***scale dependent*** effective average action $\Gamma_k[\Phi_i]$ where the scale is parameterized by k. The effective action interpolates between a microscopic UV description for the bare action at a reference scale $k = \Lambda$ and a macroscopic description at low energies described by the full quantum action, with k=0.

$$\Gamma_{k\to\Lambda} \approxeq S_{\text{bare}} \qquad \Gamma_{k\to 0} = \Gamma \tag{1.0.2}$$

The scale parameter k acts as an infrared regulator suppressing quantum fluctuations with momentum less than k. This allows us to integrate out quantum fluctuations in a controlled manner and study how models evolve with scaling.

The IR regulated generating functional is given by

$$e^{W_k[J]} \equiv Z_k[J] = \exp\left(-\Delta S_k\left[\frac{\delta}{\delta J}\right]\right) Z[J] = \int \mathcal{D}[\phi] e^{-S[\phi]-\Delta S_k[\phi]+\int J_i\phi_i} \tag{1.0.3}$$

Where

$$\Delta S_k[\phi] = \frac{1}{2}\phi_i R_k^{ij}\phi_j = \frac{1}{2}\int \frac{d^dq}{(2\pi)^d}\frac{d^dq'}{(2\pi)^d} R_k^{ab}(q,q')\phi_a(q)\phi_b(q') \tag{1.0.4}$$

With $R(q,q') \equiv R(q)\delta^d(q-q)$ acting as a mass-like regulator

$$= \frac{1}{2}\int \frac{d^dq}{(2\pi)^d} R_k^{ab}(q)\phi_a(q)\phi_b(-q) \tag{1.0.5}$$

Where the regulator function R_k must satisfy the following properties

1. $\lim_{q^2/k^2\to 0} R_k > 0$ (implements IR regularization for the path integral)
2. $\lim_{k^2/q^2\to 0} R_k = 0$ (regulator must vanish for $k \to 0$)
3. $\lim_{k\to\Lambda\to\infty} R_k = \infty$ (functional integral is then dominated by the stationary point of the action)

For this analysis we will use Litim's "optimized regulator"[2] given by

$$R_k^L(p) = \left(k^2 - p^2\right)\Theta\left(k^2 - p^2\right) \tag{1.0.6}$$

This regulator has many advantages, the most notable being that it will allow us write down analytic expressions for the β-functions.

The effective average action is then defined as a modified Legendre transform of $W[J]$

$$\Gamma_k[\Phi] = \sup_J \left(\int [-W_k[J] + \Phi_i J_i]\right) - \frac{1}{2}\Phi_i R^{ij}\Phi_j \tag{1.0.7}$$

To study the intermediate flow behavior we first define

$$t = \ln\frac{k}{\Lambda}, \qquad \partial_t = k\frac{d}{dk} \tag{1.0.8}$$

The flow of the average effective action is then described by the Wetterich equation [1]

$$\boxed{\partial_t\Gamma_k[\Phi] = \frac{1}{2}\mathrm{Tr}\left[(\Gamma_k^{(2)}[\Phi] + R_k)^{-1}\partial_t R_k\right]} \tag{1.0.9}$$

The trace should be interpreted as a basis independent way of writing down the scaling evolution.

$$\mathrm{Tr}[\hat{O}] = \sum_i \langle\Psi_i|\hat{O}|\Psi_i\rangle \tag{1.0.10}$$

Where the Ψ_i form a complete orthonormal basis of eigenstates (see Appendix A).

The flow or β-functions of masses, coupling constants, wave-function renormalization, etc. can then be obtained via a projection operator

$$\Pi_{(l,m)}\partial_t\Gamma_k[\Phi] = \frac{1}{l!}\frac{1}{m!}\partial_{\Phi_c}^l\partial_q^m\left(\partial_t\Gamma_k[\Phi_c e^{iq\cdot x}]\right)\Big|_{\Phi_c=0,q=0} \tag{1.0.11}$$

2 O(1) Scalar field theory

2.1 ϕ^4 theory β-functions in d-dimensions

The effective average action of an $O(1)$ scalar field theory with potential $V(\Phi^2)$ is given by (for a detailed calculation see Appendix B)

$$\Gamma_k[\Phi] = \int d^dx \left[\frac{1}{2}\partial^\mu\Phi(x)\partial_\mu\Phi(x) + V_k(\Phi^2)\right] \tag{2.1.1}$$

Introducing the notation $V_k'(\Phi^2) \equiv \delta V_k(\Phi^2)/\delta\Phi^2$ we have

$$\Gamma_k^{(2)}[\Phi] = -\partial^2 + 2V_k'(\Phi^2) + 4\Phi^2(x)V''(\Phi^2) \tag{2.1.2}$$

Plugging our result into the Wetterich equation we have

$$\partial_t\Gamma_k[\Phi] = \frac{1}{2}\mathrm{Tr}\left[\frac{\partial_t R_k}{[-\partial^2 + 2V_k' + 4\Phi^2(x)V'' + R_k]}\right] \tag{2.1.3}$$

The trace involves integration of position and momentum space

$$\partial_t\Gamma_k[\Phi] = \frac{1}{2}\int d^dx \int \frac{d^dp}{(2\pi)^4}\left[\frac{\partial_t R_k}{[p^2 + 2V_k' + 4\Phi^2(x)V'' + R_k]}\right] \tag{2.1.4}$$

$$= \frac{1}{(4\pi)^{d/2}\Gamma(d/2)}\int d^dx \int_0^\infty d\tilde{p}\tilde{p}^{d/2-1}\left[\frac{\partial_t R_k}{[\tilde{p}^2 + 2V_k' + 4\Phi^2(x)V'' + R_k]}\right] \tag{2.1.5}$$

Where $\Gamma(d/2)$ refers to Eulers gamma function and $\tilde{p} \equiv |p|$. Plugging in Litim's regulator noting that $\partial_t R_k^L(p) = 2k^2\Theta(k^2 - p^2)$

$$\partial_t\Gamma_k[\Phi] = \frac{2k^2}{(4\pi)^{d/2}\Gamma(d/2)}\int d^dx \int_0^{k^2} d\tilde{p}\tilde{p}^{d/2-1}\left[\frac{1}{k^2 + 2V_k' + 4\Phi^2(x)V''}\right] \tag{2.1.6}$$

$$\partial_t\Gamma_k[\Phi] = \frac{2}{d}\frac{1}{(4\pi)^{d/2}\Gamma(d/2)}\left[\frac{1}{[k^2 + 2V_k' + 4\Phi^2(x)V'']}\right] \tag{2.1.7}$$

If we consider stationary field configurations the kinetic term drops out and we are left with

$$\partial_t V_k(\Phi^2) = \frac{2}{d}\frac{1}{(4\pi)^{d/2}\Gamma(d/2)}\left[\frac{1}{[k^2 + 2V_k' + 4\Phi^2(x)V'']}\right] \tag{2.1.8}$$

To analyze the problem further we need to make an explicit choice for the potential. One choice is to take a Taylor expansion in powers of $\rho \equiv \Phi^2$

$$V(\rho) = \sum_{n=1}^{N} \lambda_{2n} \rho^n \tag{2.1.9}$$

The couplings can by extracted from the exact renormalization group flow equation via

$$\lambda_{2n} = \frac{1}{n!} \frac{\partial^n V}{\partial \rho^n}\bigg|_{\rho=0} \tag{2.1.10}$$

Likewise, the β-functions can then be obtained via

$$\beta_{2n} = \partial_t \lambda_{2n} = \frac{1}{n!} \frac{\partial^n}{\partial \rho^n} \partial_t V\bigg|_{\rho=0} \tag{2.1.11}$$

Now we redefine the couplings and fields to obtain the dimensionless β-functions

$$\bar{\Phi} = k^{\frac{2-d}{2}} \Phi \tag{2.1.12}$$

$$\bar{\lambda} = k^{(d-2)n-d} \lambda_{2n} \tag{2.1.13}$$

$$\partial_t \bar{\lambda_{2n}} = ((d-2)n - d)\bar{\lambda_{2n}} + k^{(d-2)n-d} \beta_{2n} \tag{2.1.14}$$

$$\bar{V}_k(\bar{\rho}) = k^{-d} V_k(\rho) \tag{2.1.15}$$

Eq.(2.1.8) becomes

$$\boxed{\partial_t \bar{V}_k = -d\bar{V} + (d-2)\rho \bar{V}' + \frac{2}{d} \frac{1}{(4\pi)^{d/2}\Gamma(d/2)} \left[\frac{1}{1 + 2\bar{V}_k' + 4\rho \bar{V}''}\right]} \tag{2.1.16}$$

The equation above is the basis for what is done in `scalarFRG`. Obtaining the β-functions for the dimensionless couplings is now a matter of applying Eq. (2.1.11).

2.2 Fixed points

Ultimately the β-functions allow a way to non-pertubatively analyze the UV behavior of our theory. Of particular interest are are stationary or fixed points i.e. positions in theory space in which all β-functions are zero. The trivial solution corresponding to $\lambda_2 = \lambda_4 = \cdots = \lambda_{2n} = 0$ is called a Gaussian fixed point. These fixed points correspond to a free theory with no interactions. Non-trivial solutions, corresponding to renormalizable interacting theories, are called Wilson-Fisher fixed points and are the main motivation in what follows.

3 scalarFRG

The main goals of this program are to

1. Compute the $\bar{\beta}$-functions for a d-dimensional O(1) scalar field theory for an nth order potential using the `SymPy` computer algebra package
2. Given the $\bar{\beta}$-functions compute Wilson-Fisher fixed points
3. Create theory space flow-plots

3.1 Root-finding

Finding Wilson-Fisher fixed points amounts to finding the simultaneous roots of the $\bar{\beta}$-functions. Depending on the truncation order we end up with a system of coupled non-linear equations of the form

$$\begin{aligned} \bar{\beta}_2(\bar{\lambda}_2, \bar{\lambda}_4, \cdots, \bar{\lambda}_{2n}) &= 0 \\ \bar{\beta}_4(\bar{\lambda}_2, \bar{\lambda}_4, \cdots, \bar{\lambda}_{2n}) &= 0 \\ &\vdots \\ \bar{\beta}_{2n}(\bar{\lambda}_2, \bar{\lambda}_4, \cdots, \bar{\lambda}_{2n}) &= 0 \end{aligned} \tag{3.1.1}$$

Defining $\mathbf{\Lambda} \equiv (\bar{\lambda}_2, \bar{\lambda}_4, \cdots, \bar{\lambda}_{2n})$ and $\mathbf{B} = (\bar{\beta}_2, \bar{\beta}_4, \cdots, \bar{\beta}_n)$ we can write the above system of equations as

$$\mathbf{B}(\mathbf{\Lambda}) = \mathbf{0} \tag{3.1.2}$$

Solving this problem numerically can be broken up into two parts: 1. Numerically finding roots and 2. Solving system of equations

3.1.1 Newton-Raphson

The Newton-Raphson method is a simple and fast algorithm for finding roots of simultaneous equations. We start with the series expansion of $\beta_i(\mathbf{\Lambda})$ about the point $\mathbf{\Lambda}$

$$\beta_i(\mathbf{\Lambda} + \Delta\mathbf{\Lambda}) = \beta_i(\mathbf{\Lambda}) + \sum_{j=1}^{n} \frac{\partial \beta_i}{\partial \lambda_j} \Delta\lambda_j + \mathcal{O}(\Delta\lambda^2) \tag{3.1.3}$$

The above can be written compactly as

$$\mathbf{B}(\mathbf{\Lambda} + \Delta\mathbf{\Lambda}) = \mathbf{B}(\mathbf{\Lambda}) + \mathbf{J}(\mathbf{\Lambda})\Delta\mathbf{\Lambda} \tag{3.1.4}$$

Where $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J}_{ij} = \frac{\partial \beta_i}{\partial \lambda_j} \tag{3.1.5}$$

If we consider $\mathbf{\Lambda}$ to be the current approximation of a fixed point and $\mathbf{\Lambda} + \Delta\mathbf{\Lambda}$ as the improved solution we can set $\mathbf{B}(\mathbf{\Lambda} + \Delta\mathbf{\Lambda}) = 0$ and solve for the correction $\Delta\mathbf{\Lambda}$. We find

$$\mathbf{J}\Delta\mathbf{\Lambda} = -\mathbf{B}(\mathbf{\Lambda}) \tag{3.1.6}$$

We obtain a set of linear equations for $\Delta\mathbf{\Lambda}$ which must be solved for $\Delta\mathbf{\Lambda}$. The numerical method I chose to implement was Gauss elimination.

3.1.2 Gauss elimination

The Gauss-elimination algorithm is also decomposed into two steps: 1. The elimination phase in which we convert the Jacobian matrix $\mathbf{J}$ into an upper-diagonal matrix and 2. the back-substitution phase where we plug our results back in and solve for $\Delta\mathbf{\Lambda}$ [3].

1. Elimination stage: If we consider an $n \times n$ system of equations i.e.

$$\begin{pmatrix} M_{11} & \cdots & M_{1k} & \cdots & M_{1n} & v_1 \\ M_{21} & \ddots & & & & \vdots \\ \vdots & & M_{kk} & & & \vdots \\ \vdots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & \vdots \\ M_{n1} & \cdots & M_{nk} & \cdots & M_{nn} & v_n \end{pmatrix} \tag{3.1.7}$$

   The elimination stage proceeds by iterating through "pivot" rows. The first non-zero element in the kth pivot row begins on the diagonal M_{kk}. The rows above the pivot have already been placed in upper diagonal form. The algorithm proceeds by eliminating the M_{kk}th variable from each of the remaining rows below the pivot row. The row transformations are given by

$$M_{ij} \to M_{ij} - \kappa M_{kj} \qquad j = k, k+1, \cdots, n \tag{3.1.8}$$

$$v_i \to v_i - \kappa v_k \tag{3.1.9}$$

   Where κ is chosen to eliminate the M_{kk} variable from the jth row i.e. $\kappa = M_{ik}/M_{kk}$.

2. Back substitution: At this point in the algorithm we are left with a system of the form

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} & \cdots & M_{1n} & v_1 \\ 0 & M_{22} & M_{23} & \cdots & M_{2n} & v_2 \\ 0 & 0 & M_{33} & \cdots & M_{3n} & v_3 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & M_{nn} & v_n \end{pmatrix} \tag{3.1.10}$$

The final row is given by the equation

$$M_{nn}x_n = v_n \rightsquigarrow x_n = \frac{v_n}{M_{nn}} \tag{3.1.11}$$

This value of x_n can now be be placed into the $(n-1)$th row

$$M_{(n-1)(n-1)}x_{n-1} + M_{(n-1)n}x_n = v_{n-1} \tag{3.1.12}$$

$$x_{n-1} = \frac{v_{n-1} - M_{(n-1)n}\dfrac{v_n}{M_{nn}}}{M_{(n-1)(n-1)}} \tag{3.1.13}$$

This can be done iteratively until we have the solution for x_1. To summarize, x_k from the kth equation (row)

$$M_{kk}x_k + M_{k,k+1}x_{k+1} + \cdots + M_{kn}x_n = v_k \tag{3.1.14}$$

Is given by

$$x_k = \left(v_k - \sum_{j=k+1}^{n} M_{kj}x_j\right)\frac{1}{M_{kk}}, \qquad k = n-1, n-2, \cdots, 1 \tag{3.1.15}$$

When the dust settles we are left with the solution for the correction vector $\Delta\mathbf{\Lambda}$ describing the corrections to λ_i that bring us closer to the true solution of the root equations. These corrections are then added to λ. This process can be repeated until the change $\Delta\mathbf{\Lambda}$ reaches some pre-assigned tolerance or over a prescribed number of iterations. At the end of the day we are left with a vector of coupling values which correspond to a fixed point in our theory.

The algorithm can be summarized in the following steps:

1. Provide a guess for the solution vector $\mathbf{\Lambda}$

2. Evaluate $\mathbf{B}(\mathbf{\Lambda})$
3. Compute the Jacobian $\mathbf{J}(\mathbf{\Lambda})$
4. Set up and solve the simultaneous equations for the corrections $\Delta\mathbf{\Lambda}$
5. Set $\mathbf{\Lambda} \to \mathbf{\Lambda} + \Delta\mathbf{\Lambda}$ and repeat steps 2-5

There are two obvious pitfalls of the Newton-Raphson method.

1. A solution is not guaranteed. In order for convergence we need to provide a "good" initial guess. This is especially a problem when dealing with systems of equations which have no obvious initial solutions (like the β-functions).
2. The algorithm relies on the knowledge of the user-provided Jacobian matrix. This especially becomes a problem when dealing with a large system of equations where analytic computation of the derivatives is impractical.

The first issue can be solved by doing an initial search for roots by varying each parameter individually and seeing when $\mathbf{B}(\mathbf{\Lambda})$ switches from being positive valued to negative valued or vice versa indicating a root. This allows the user to have no knowledge of the solution and still obtain convergence. In `scalarFRG` I have not been able to implement this yet, however it is next on my list.

The typical way to solve the second issue is by computing the Jacobian matrix with the finite difference approximation. I have gone a different route and used `SymPy` to compute the Jacobian matrix analytically.

3.1.3 `newton_raphson(f, x, guess, N = 30, tol = 10e-8)`

I have implemented the above algorithm as `newton_raphson` within `scalarFRG`. The arguments of the method are as follows: `f`: list of strings **in `SymPy` syntax** containing the system of equations, `x`: list of strings containing the independent variables of interest, and `guess`: list of floats containing the initial guess. For the optional arguments we have `N`: number of iterations, set to 30 as default and `tol`: tolerance of corrections used to signal termination of the algorithm.

The efficacy of my algorithm was proven by comparing to Mathematica's `NSolve` function. For example the system of equations

$$\begin{aligned} &\sin x + y^2 + \ln z - 7 = 0 \\ &3x + 2^y - z^3 + 1 = 0 \\ &x + y + z - 5 = 0 \end{aligned} \tag{3.1.16}$$

was solved using `newton_raphson()`

```
f1 = "sin(x) + y**2 + ln(z) - 7"

f2 = "3*x + 2**y - z**3 + 1"

f2 = "x+y+z-5"

root = newton_raphson([f1,f2,f3],['x','y','z'],[1.0,1.0,1.0])
```

With the result

```
x,y,z = [0.59905376 2.3959314 2.00501484]
```

Mathematica's `NSolve` gives the indistinguishable result

```
x,y,z = [0.59905376 2.3959314 2.00501484]
```

NOTE: I have also implemented two other root searching algorithms (bisection and secant) as well as a numerical ODE solver (4th order Runge-Kutta) which I have neglected to save time. Ultimately, I did not use these methods in this version of `scalarFRG`. In principle I could use the bisection or secant methods as a security blanket in the case for which the Newton-Raphson method doesn't converge. This will be investigated in the future. The ODE solver will prove useful if I ever need to solve for the exact flow for two given initial coupling values. For the general analysis of the scale evolution of a theory it is better to use β-function flow plots and fixed points as they allow you to determine the asymptotic nature of your theory over the whole coupling parameter space.

3.2 `class O1(d, n)`

The `O1` class contains methods which implement the three goal stated at the beginning of this section. The class takes two arguments in its constructor, `d` corresponds to the number of dimensions the user would like to compute the β-functions in and `n` corresponds to the order of the potential the user would like to expand to. For example, `O1(1,2)` corresponds to a scalar field theory in 1-dimension with a potential of the form $V(\rho) = \lambda_2\rho + \lambda_4\rho^2$.

The constructor takes these arguments and using Eq.(2.1.16) computes the dimensionless β-functions for all `n` couplings using `SymPy`. The β-functions are stored as class variables.

3.2.1 flow_plot(lambdax, lambday, lamxcenter = 0, lamycenter = 0, w=1.5, **kwargs)

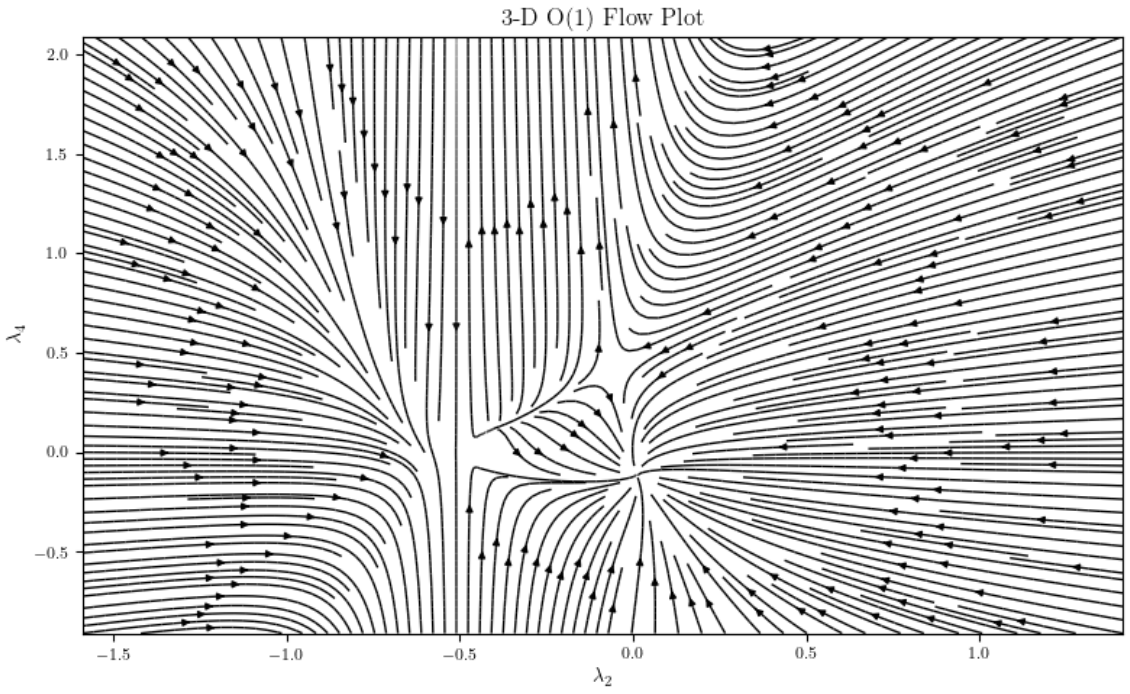
β-function flow plots are extremely useful visual tools in understanding the flow of parameters in theory space. The plots also allow a better way of seeing and understanding the flow of parameters around fixed points of a theory. To create these plots I have utilized `MatPlotLib`'s `streamplot()` function.

As arguments the user needs to provide two strings corresponding to flow parameters (i.e. `lambda2, lambda4, ...`) the first argument will be plotted on the x-axis and the second argument will be plotted on the y-axis. The user can choose to plot about the center point `lamxcenter, lamycenter` extending a length `w` on all sides. This may be useful when one wants to plot near a fixed point solution. If the potential has been chosen such that `n`>2 the user must specify the values for all other couplings which are not being plotted as `**kwargs`.

For example,

```
O1(3,4).flow_plot('lambda2','lambda4',1.235,2.045,lambda6=0.3,lambda8=0.2)
```

Produces the following flow plot in which the arrows point from the IR $\rightarrow$ UV.



When the method is ran, a new folder called `O1_figs` is created in the current working directory. The figure in this folder under the name `flow_d-n-.png`. Where the dashes are replaced by the class variables `d` and `n`.

3.2.2 `fixed_pts(guess)`

The `fixed_pts()` method takes a list of floats as an argument which is then used as the initial guess for the Newton-Raphson algorithm outlined in section 3.1. The method returns a single fixed point of the theory (assuming a "good" guess has been provided). Currently `scalarFRG` relies heavily on a good guess provided by the user, in future iterations I would like to implement the root search algorithm described at the end of section 3.1 to relieve this burden. Another subtlety is that there may be more than one Wilson-Fisher fixed point. Finding numerous roots to the β-functions could also be obtained with the described search algorithm.

3.2.3 `texput()`

The intended function of the `texput` method is to provide a summary of the results given by the `O1` class to a LaTeX file. This includes output of the chosen potential, the calculated β-functions, fixed points of the theory, and flow plots around the fixed points of the theory. Currently this feature is only partially implemented.

4 Outlook

In its current form `scalarFRG` provides a package for computing β-functions, fixed points, and flow plots for a d-dimensional O(1) scalar field theory. In the future it would be interesting to implement, in no particular order, the following:

1. Calculation of critical exponents to label fixed points as attractive or repulsive
2. Implementing an adaptive root search algorithm which will be able to find all fixed point solutions
3. Using `SymPy` to compute the flow equations given an effective average action. For example I could feed in the effective average action given in Eq. (2.1.1) and return Eq. (2.1.16)
4. Generalizing to d-dimensional $O(N)$ scalar field theories
5. Using different approximation schemes

6. Investigating different regulator function which then may involve numerical integration of the flow equations
7. Investigating three-dimensional flow plots/identifying flow features
8. Performing the same analysis but for QCD
9. Performing the same analysis but for BSM physics

References

[1] Wetterich, Christof. "Exact evolution equation for the effective potential." Physics Letters B 301.1 (1993): 90-94.

[2] Litim, Daniel F. "Optimized renormalization group flows." Physical Review D 64.10 (2001): 105007., Litim, Daniel F. "Optimisation of the exact renormalisation group." Physics Letters B 486.1-2 (2000): 92-99.

[3] Kiusalaas, Jaan. Numerical methods in engineering with Python 3. Cambridge university press, 2013.

5 Appendix A

To express in a particular representation we insert the complete set of states, for example in position space the trace is written as

$$\mathrm{Tr}[\hat{O}] = \sum_i \langle \Psi_i | \hat{O} | \Psi_i \rangle = \sum_i \int d^d x d^d y \, \langle \Psi_i | x \rangle \, \langle x | \hat{O} | y \rangle \, \langle y | \Psi_i \rangle \tag{5.0.1}$$

$$= \int d^d x d^d y \, \langle x | \hat{O} | y \rangle \sum_i \Psi_i^*(x) \Psi_i(y) = \int d^d x d^d y \, \langle x | \hat{O} | y \rangle \, \delta^d(x-y) = \int d^d x \, \langle x | \hat{O} | x \rangle \tag{5.0.2}$$

$$\equiv \int d^d x \hat{O}(x) \tag{5.0.3}$$

Sometimes it is convenient to work in a momentum basis, performing a Fourier transform from the position-basis

$$\int d^d x \, \langle x | \hat{O} | x \rangle = \int d^d x d^d p d^d p' \, \langle x | p \rangle \, \langle p | \hat{O} | p' \rangle \, \langle p' | x \rangle \tag{5.0.4}$$

$$= \frac{1}{(2\pi)^d} \int d^d x d^d p d^d p' e^{i(p-p')\cdot x} \, \langle p | \hat{O} | p' \rangle \tag{5.0.5}$$

$$= \frac{1}{(2\pi)^d} \int d^d p d^d p' (2\pi)^d \delta^d(p-p') \, \langle p | \hat{O} | p' \rangle = \int d^d p \, \langle p | \hat{O} | p \rangle \tag{5.0.6}$$

$$\equiv \int d^d p \hat{O}(p) \tag{5.0.7}$$

6 Appendix B

Truncating the potential at $n = 2$ we have the following effective average action

$$\Gamma_k[\Phi] = \int d^d x \left[\frac{1}{2} z_k g_{\mu\nu} \partial^\mu \Phi(x) \partial^\nu \Phi(x) + \frac{1}{2} m_k^2 \Phi^2(x) + \frac{\lambda_k}{4!} \Phi^4(x)\right] \tag{6.0.1}$$

Taking a scale derivative of the average effective action yields

$$\partial_t \Gamma_k[\phi] = \int d^d x \left[\frac{1}{2} (\partial_t z_k) \partial^\mu \Phi(x) \partial_\mu \Phi(x) + \frac{1}{2} (\partial_t m_k^2) \Phi^2(x) + \frac{(\partial_t \lambda_k)}{4!} \Phi^4(x)\right] \tag{6.0.2}$$

We can project out the coupling β-functions via the projecting operator

$$\Pi_{(l,m)} \partial_t \Gamma_k[\Phi] = \frac{1}{l!} \frac{1}{m!} \partial_{\Phi_c}^l \partial_q^m \left(\partial_t \Gamma_k[\Phi_c e^{iq\cdot x}]\right) \Big|_{\Phi_c=0, q=0} \tag{6.0.3}$$

For example, to project out $\partial_t z_k$ we will need $\Pi_{(2,2)}$ going term by term for explicitness

$$- \int d^d x \frac{1}{2!} \frac{1}{2!} \frac{1}{2} \partial_t z_k \partial_{\Phi_c}^2 \partial_q^2 \left[\partial_\mu (\Phi_c e^{iq\cdot x}) \partial^\mu (\Phi_c e^{iq\cdot x})\right] \Big|_{\Phi_c=0, q=0} \tag{6.0.4}$$

$$= \frac{-(i)^2}{8} \int d^d x \partial_t z_k \partial_{\Phi_c}^2 \partial_q^2 \left[q^2 \Phi_c^2 e^{2iq\cdot x}\right] \Big|_{\Phi_c=0, q=0} \tag{6.0.5}$$

$$= \frac{1}{8} \int d^d x \partial_t z_k \partial_{\Phi_c}^2 \left[2\Phi_c^2 e^{2iq\cdot x} + 8ixq\Phi_c^2 e^{2iq\cdot x} + (2ix)^2 q^2 \Phi_c^2 e^{2iq\cdot x}\right] \Big|_{\Phi_c=0, q=0} \tag{6.0.6}$$

$$= \frac{1}{8} \int d^d x \partial_t z_k \partial_{\Phi_c}^2 \left[2\Phi_c^2\right] \Big|_{\Phi_c=0} \tag{6.0.7}$$

$$= \frac{1}{2} \int d^d x \partial_t z_k \tag{6.0.8}$$

For the next term we have

$$\int d^d x \frac{1}{2!} \frac{1}{2!} \frac{1}{2} \partial_t m_k^2 \partial_{\Phi_c}^2 \partial_q^2 \left[(\Phi_c^2 e^{2iq\cdot x})\right] \Big|_{\Phi_c=0, q=0} = \frac{1}{8} \int d^d x \partial_t m_k^2 \partial_{\Phi_c}^2 [\Phi_c^2 (2ix)^2] \Big|_{\Phi_c=0} \tag{6.0.9}$$

$$= - \int d^d x x^2 \partial_t m_k^2 \tag{6.0.10}$$

And finally for the last term

$$\int d^d x \frac{1}{2!} \frac{1}{2!} \frac{1}{4!} \partial_t \lambda_k \partial_q^2 \partial_{\Phi_c}^2 \left[\Phi_c^4 e^{4iq\cdot x}\right] \Big|_{\Phi_c=0, q=0} \tag{6.0.11}$$

$$= \frac{1}{96}\int d^dx \partial_t \lambda_k \partial_q^2 \left[(4\cdot 3)\Phi_c^2 e^{4iq\cdot x}\right]\Big|_{\Phi_c=0,q=0} = 0 \tag{6.0.12}$$

Putting it all together we find

$$\Pi_{(2,2)}\partial_t\Gamma_k[\Phi] = \int d^dx \left[\frac{1}{2}\partial_t z_k - x^2\partial_t m_k^2\right] \tag{6.0.13}$$

The other parameters can be found easily by acting with $\Pi_{(2,0)}$ and $\Pi_{(4,0)}$, we are left with the following three results

$$\boxed{\Pi_{(2,2)}\partial_t\Gamma_k[\Phi] = \int d^dx \left[\frac{1}{2}\partial_t z_k - x^2\partial_t m_k^2\right]} \tag{6.0.14}$$

$$\boxed{\Pi_{(2,0)}\partial_t\Gamma_k[\Phi] = \frac{1}{2}\int d^dx \partial_t m_k^2} \tag{6.0.15}$$

$$\boxed{\Pi_{(4,0)}\partial_t\Gamma_k[\Phi] = \frac{1}{4!}\int d^dx \partial_t \lambda_k} \tag{6.0.16}$$

Now we look to the RHS of the Wetterich equation

$$\frac{1}{2}\mathrm{Tr}\left[(\Gamma_k^{(2)}[\Phi] + R_k)^{-1}\partial_t R_k\right] = \tag{6.0.17}$$

First we need to compute $\Gamma_k^{(2)}[\Phi]$

$$\frac{\delta^2\Gamma_k}{\delta\Phi(y)\delta\Phi(z)} = \int d^dx \Bigg\{ \frac{\delta^2}{\delta\Phi(y)\delta\Phi(z)}\left[\frac{1}{2}z_k\partial^\mu\Phi(x)\partial_\mu\Phi(x)\right] + \frac{\delta^2}{\delta\Phi(y)\delta\Phi(z)}\left[\frac{1}{2}m_k^2\Phi^2(x)\right] + \frac{\delta^2}{\delta\Phi(y)\delta\Phi(z)}\left[\frac{\lambda_k}{4!}\Phi^4(x)\right]\Bigg\} \tag{6.0.18}$$

$$= \int d^dx \left\{\frac{1}{2}\frac{\delta^2}{\delta\Phi(y)\delta\Phi(z)}\left[\Phi(x)\left(-z_k\partial^2 + m_k^2\right)\Phi(x)\right] + \frac{1}{4!}\lambda_k\frac{\delta^2}{\delta\Phi(y)\delta\Phi(z)}\Phi^4(x)\right\} \tag{6.0.19}$$

For the first term I have integrated by parts (with the assumption that our field $\Phi(x)$ vanishes at the boundaries),

$$\int d^dx[\partial^\mu\Phi\partial_\mu\Phi] = -\int d^dx\Phi\partial^2\Phi \tag{6.0.20}$$

Defining $\Delta \equiv -z_k\partial^2 + m_k^2$ the first term becomes

$$\frac{1}{2}\frac{\delta}{\delta\Phi(y)}\left[\frac{\delta}{\delta\Phi(z)}[\Phi(x)]\Delta\Phi(x) + \Phi(x)\Delta\frac{\delta}{\delta\Phi(z)}[\Phi(x)]\right] \tag{6.0.21}$$

$$= \frac{1}{2}\left[\Delta\frac{\delta}{\delta\Phi(y)}\Phi(z) + \frac{\delta}{\delta\Phi(y)}\Phi(z)\Delta\right] \tag{6.0.22}$$

$$= \Delta \tag{6.0.23}$$

The second term gives

$$\frac{1}{4!}\lambda_k\frac{\delta^2}{\delta\Phi(y)\delta\Phi(z)}\Phi^4(x) = \frac{4\cdot 3}{4!}\lambda_k\Phi^2(x) = \frac{1}{2}\lambda_k\Phi^2(x) \tag{6.0.24}$$

Putting this all together we are left with

$$\boxed{\Gamma_k^{(2)}[\Phi] = -z_k\partial^2 + m_k^2 + \frac{1}{2}\lambda_k\Phi^2(x)} \tag{6.0.25}$$

Now we can plug this into the scale evolution Eq.(1.0.9)

$$\partial_t\Gamma_k[\Phi] = \frac{1}{2}\mathrm{Tr}\left[\frac{1}{\left[-z_k\partial^2 + m_k^2 + \frac{\lambda_k}{2}\Phi^2(x) + R_k\right]}\partial_t R_k\right] \tag{6.0.26}$$

Now we define $\tilde{\Delta} \equiv -z_k\partial^2 + m_k^2 + R_k$

$$= \frac{1}{2}\mathrm{Tr}\left[\frac{1}{\tilde{\Delta}\left[\mathbb{1} + \frac{1}{2}\tilde{\Delta}^{-1}\lambda_k\Phi^2(x)\right]}\partial_t R_k\right] \tag{6.0.27}$$

Now I can make use of the following operator expansion[1]

$$\left[\mathbb{1} + \tilde{\Delta}^{-1}\lambda_k\Phi^2(x)\right]^{-1} = \sum_{k=0}^{\infty}\frac{(-1)^k}{2^k}\left[\tilde{\Delta}^{-1}\lambda_k\Phi^2(x)\right]^k \tag{6.0.28}$$

[1]For this expansion to be valid the eigenvalues λ_i of the operator $\tilde{\Delta}^{-1}\lambda_k\Phi^2$ must satisfy the condition $|\lambda_i| < 1$.

Truncating all operators above Φ^4 we are left with

$$= \frac{1}{2}\mathrm{Tr}\left[\tilde{\Delta}^{-1}\left\{\mathbb{1} - \frac{1}{2}\tilde{\Delta}^{-1}\lambda_k\Phi^2(x) + \frac{1}{4}\left(\tilde{\Delta}^{-1}\lambda_k\Phi^2(x)\right)^2 + \mathcal{O}(\Phi^6)\right\}\partial_t R_k\right] \tag{6.0.29}$$

$$= \frac{1}{2}\mathrm{Tr}\left[\left\{\tilde{\Delta}^{-1} - \frac{1}{2}(\tilde{\Delta}^{-1})^2\lambda_k\Phi^2(x) + \frac{1}{4}(\tilde{\Delta}^{-1})^3\lambda_k^2\Phi^4(x)\right\}\partial_t R_k\right] \tag{6.0.30}$$

Expanding the trace in momentum space ($\partial^2 \rightsquigarrow i^2p^2 = -p^2$)

$$\begin{aligned}\partial_t\Gamma_k[\Phi] = \frac{1}{2}\frac{1}{(2\pi)^d}\int d^dx\int d^dp\Big[\Big\{&\left(z_kp^2 + m_k^2 + R_k(p)\right)^{-1}\\ &- \frac{\lambda_k}{2}\left(z_kp^2 + m_k^2 + R_k(p)\right)^{-2}\langle p|\Phi\Phi|p\rangle\\ &+ \frac{\lambda_k^2}{4}\left(z_kp^2 + m_k^2 + R_k(p)\right)^{-3}\langle p|\Phi\Phi\Phi\Phi|p\rangle\Big\}\partial_t R_k\Big]\end{aligned} \tag{6.0.31}$$

$$\begin{aligned}= \frac{1}{2}\frac{1}{(2\pi)^d}\int d^dx\int d^dp\Big[\Big\{&\left(z_kp^2 + m_k^2 + R_k(p)\right)^{-1}\\ &- \frac{\lambda_k}{2}\left(z_kp^2 + m_k^2 + R_k(p)\right)^{-2}\Phi^2(p)\\ &+ \frac{\lambda_k^2}{4}\left(z_kp^2 + m_k^2 + R_k(p)\right)^{-3}\Phi^4(p)\Big\}\partial_t R_k\Big]\end{aligned} \tag{6.0.32}$$

Applying the same projection operators (Eq. (6.0.3)) to this expression we are left with

$$\boxed{\Pi_{(2,2)}\partial_t\Gamma_k[\Phi] = \frac{1}{(2\pi)^d}\int d^dx\int d^dp\frac{\lambda_k}{2}\left[\left(z_kp^2 + m_k^2 + R_k\right)^{-2}x^2\partial_t R_k\right]} \tag{6.0.33}$$

$$\boxed{\Pi_{(2,0)}\partial_t\Gamma_k[\Phi] = -\frac{1}{(2\pi)^d}\int d^dx\int d^dp\frac{\lambda_k}{4}\left[(z_kp^2 + m_k^2 + R_k)^{-2}\partial_t R_k\right]} \tag{6.0.34}$$

$$\boxed{\Pi_{(4,0)}\partial_t\Gamma_k[\Phi] = \frac{1}{(2\pi)^d}\int d^dx\int d^dp\frac{\lambda_k^2}{8}\left[(z_kp^2 + m_k^2 + R_k)^{-3}\partial_t R_k\right]} \tag{6.0.35}$$

Equating with Eqs.(6.0.14)(6.0.15)(6.0.16) we find the following three relations

$$\frac{1}{2}\partial_t z_k - x^2\partial_t m_k^2 = \frac{1}{(2\pi)^d}\int d^dp\frac{\lambda_k}{2}\left[\left(z_kp^2 + m_k^2 + R_k\right)^{-2}x^2\partial_t R_k\right] \tag{6.0.36}$$

$$\frac{1}{2}\partial_t m_k^2 = -\frac{1}{(2\pi)^d}\int d^d p \frac{\lambda_k}{4}\left[(z_k p^2 + m_k^2 + R_k)^{-2}\partial_t R_k\right] \tag{6.0.37}$$

$$\frac{1}{4!}\partial_t \lambda_k = \frac{1}{(2\pi)^d}\int d^d p \frac{\lambda_k^2}{8}\left[(z_k p^2 + m_k^2 + R_k)^{-3}\partial_t R_k\right] \tag{6.0.38}$$

Plugging the result for $\partial_t m_k$ in Eq.(6.0.34) into Eq.(6.0.33) we see that

$$\boxed{\partial_t z_k = 0} \tag{6.0.39}$$

To make further progress we need to choose a regulator function R_k. To start we will look at the optimized Litim regulator [2]

$$R_k^L(p) = z_k\left(k^2 - p^2\right)\Theta\left(k^2 - p^2\right) \tag{6.0.40}$$

Where $\Theta(x)$ is the Heaviside step function. Thus,

$$\partial_t R_k^L = 2 z_k k^2 \Theta(k^2 - p^2) \tag{6.0.41}$$

Plugging this regulator back in to Eq.(6.0.37)

$$\partial_t m_k^2 = -\frac{\lambda_k}{2}\frac{1}{(2\pi)^d}\int d^d p\left[\left(z_k p^2 + m_k^2 + z_k(k^2 - p^2)\Theta(k^2 - p^2)\right)^{-2} 2 z_k k^2 \Theta(k^2 - p^2)\right] \tag{6.0.42}$$

$$= -\frac{\lambda_k}{(2\pi)^d}\int_{p^2<k^2} d^d p\left[z_k p^2 + m_k^2 + z_k(k^2 - p^2)\right]^{-2} z_k k^2 \tag{6.0.43}$$

$$= -\frac{\lambda_k}{(2\pi)^d}\int_{p^2<k^2} d^d p\left[\frac{z_k k^2}{(m_k^2 + z_k k^2)^2}\right] \tag{6.0.44}$$

Now the problem has been boiled down to computing the volume of d-dimensional sphere of radius k

$$\Omega_d(k) = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} k^d \tag{6.0.45}$$

Where $\Gamma(x) = (x-1)!$ is the Euler gamma-function. Note

$$\Gamma(n + \frac{1}{2}) = \frac{(2n)!}{4^n n!}\sqrt{\pi} \qquad \text{for } n \in \mathbb{Z}^* \tag{6.0.46}$$

Thus we find

$$= -\frac{z_k \lambda_k}{(2\pi)^d}\left[\frac{k^2}{(m_k^2 + z_k k^2)^2}\right]\frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} k^d \tag{6.0.47}$$

Our second flow equation is then given by

$$\boxed{\partial_t m_k^2 = \frac{-\lambda_k z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)} \frac{k^{2+d}}{(m_k^2 + z_k k^2)^2}} \tag{6.0.48}$$

Plugging the regulator into Eq. (6.0.38) gives

$$\partial_t \lambda_k = \frac{3\lambda_k^2}{(2\pi)^d} \int d^d p \left[\left(z_k p^2 + m_k^2 + z_k (k^2 - p^2)\Theta(k^2 - p^2) \right)^{-3} 2 z_k k^2 \Theta(k^2 - p^2) \right] \tag{6.0.49}$$

$$= \frac{6\lambda_k^2}{(2\pi)^d} \int_{p^2<k^2} d^d p \left[\left(z_k p^2 + m_k^2 + z_k (k^2 - p^2) \right)^{-3} z_k k^2 \right] \tag{6.0.50}$$

$$= \frac{6\lambda_k^2}{(2\pi)^d} \int_{p^2<k^2} d^d p \left[\frac{z_k k^2}{\left(m_k^2 + z_k k^2 \right)^3} \right] \tag{6.0.51}$$

We are left with our final flow equation

$$\boxed{\partial_t \lambda_k = \frac{6\lambda_k^2 z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)} \frac{k^{2+d}}{(m_k^2 + z_k k^2)^3}} \tag{6.0.52}$$

We can perform a redefinition of the couplings to obtain dimensionless β-functions

$$\bar{\lambda}_k = k^{d-4} \lambda_k \longrightarrow \lambda_k = \bar{\lambda}_k k^{4-d} \tag{6.0.53}$$

$$\bar{m}_k^2 = k^{-2} m_k^2 \longrightarrow m_k^2 = \bar{m}_k^2 k^2 \tag{6.0.54}$$

We have

$$\partial_t \lambda_k = k \frac{d}{dk} (\bar{\lambda}_k k^{4-d}) = k \left((4-d) k^{3-d} \bar{\lambda}_k + k^{4-d} \frac{d}{dk} \bar{\lambda}_k \right) = k^4 k^{-d} \left((4-d) \bar{\lambda}_k + k \frac{d}{dk} \bar{\lambda}_k \right) \tag{6.0.55}$$

$$\frac{6\lambda_k^2 z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)} \frac{k^{2+d}}{(m_k^2 + z_k k^2)^3} = \frac{6\bar{\lambda}_k^2 k^{8-2d} k^{2+d} z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right) (\bar{m}_k^2 k^2 + z_k k^2)^3} \tag{6.0.56}$$

$$= \frac{6\bar{\lambda}_k^2 k^4 k^{-d} z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right) (\bar{m}_k^2 + z_k)^3} \tag{6.0.57}$$

Equating both sides leaves us with the first dimensionless flow of interest

$$\boxed{\partial_t \bar{\lambda}_k = \frac{6\bar{\lambda}_k^2 z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)(\bar{m}_k^2 + z_k)^3} - (4-d)\bar{\lambda}_k} \tag{6.0.58}$$

We also have

$$\partial_t m_k^2 = k\frac{d}{dk}(\bar{m}_k^2 k^2) = k\left(2k\bar{m}_k^2 + k^2\frac{d}{dk}\bar{m}_k^2\right) \tag{6.0.59}$$

$$\frac{-\lambda_k z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)} \frac{k^{2+d}}{(m_k^2 + z_k k^2)^2} = \frac{-\bar{\lambda}_k k^{4-d} z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)} \frac{k^{2+d}}{(\bar{m}_k^2 k^2 + z_k k^2)^2} \tag{6.0.60}$$

$$= \frac{-\bar{\lambda}_k k^2 z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)(\bar{m}_k^2 + z_k)^2} \tag{6.0.61}$$

Equating both sides leaves us with

$$\boxed{\partial_t \bar{m}_k^2 = \frac{-\bar{\lambda}_k z_k}{(2\sqrt{\pi})^d \Gamma\left(\frac{d}{2}+1\right)(\bar{m}_k^2 + z_k)^2} - 2\bar{m}_k^2} \tag{6.0.62}$$